الگوریتم‌های پیشرفته

کلاس حل تمرین

درخت جستجوی دودویی

تیم دستیاران آموزشی

مقدمه

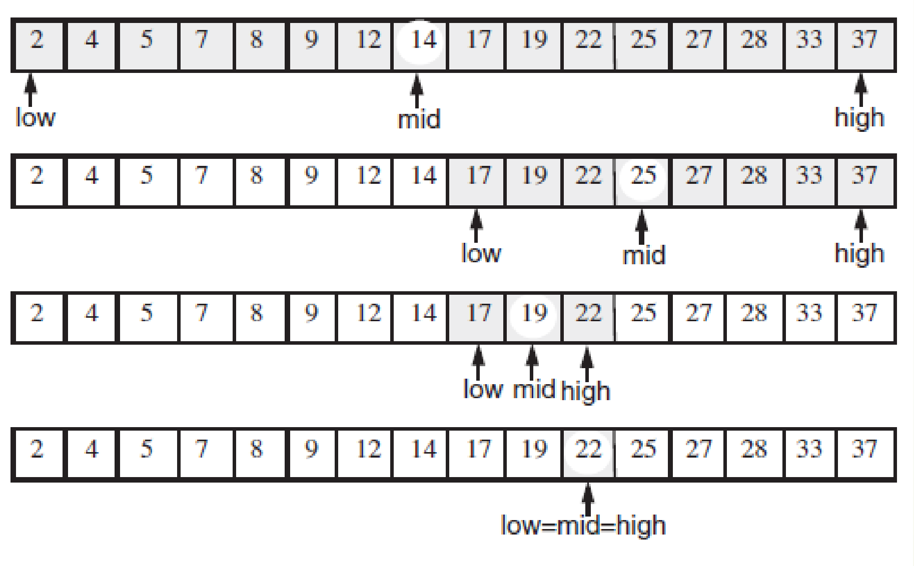
الگوریتم جستجوی دودویی

الگوریتم

```
Algorithm BinarySearch(A, k, low, high):
  Input: An ordered array, A, storing n items, whose keys are accessed with
    method key(i) and whose elements are accessed with method elem(i); a
    search key k; and integers low and high
  Output: An element of A with key k and index between low and high, if such
    an element exists, and otherwise the special element null

  if low > high then
     return null
  else
     mid ← ⌊(low + high)/2⌋
     if k = key(mid) then
        return elem(mid)
     else if k < key(mid) then
        return BinarySearch(A, k, low, mid − 1)
     else
        return BinarySearch(A, k, mid + 1, high)
```

نمونه اجرای الگوریتم



الگوریتم جستجوی دودویی (ادامه) – تحلیل الگوریتم

- تعداد عناصر باقی‌مانده بعد از هر بار اجرای الگوریتم:

$$(\mathrm{mid}-1)-\mathrm{low}+1=\left\lfloor\frac{\mathrm{low}+\mathrm{high}}{2}\right\rfloor-\mathrm{low}\le\frac{\mathrm{high}-\mathrm{low}+1}{2}$$

$$\mathrm{high}-(\mathrm{mid}+1)+1=\mathrm{high}-\left\lfloor\frac{\mathrm{low}+\mathrm{high}}{2}\right\rfloor\le\frac{\mathrm{high}-\mathrm{low}+1}{2}$$

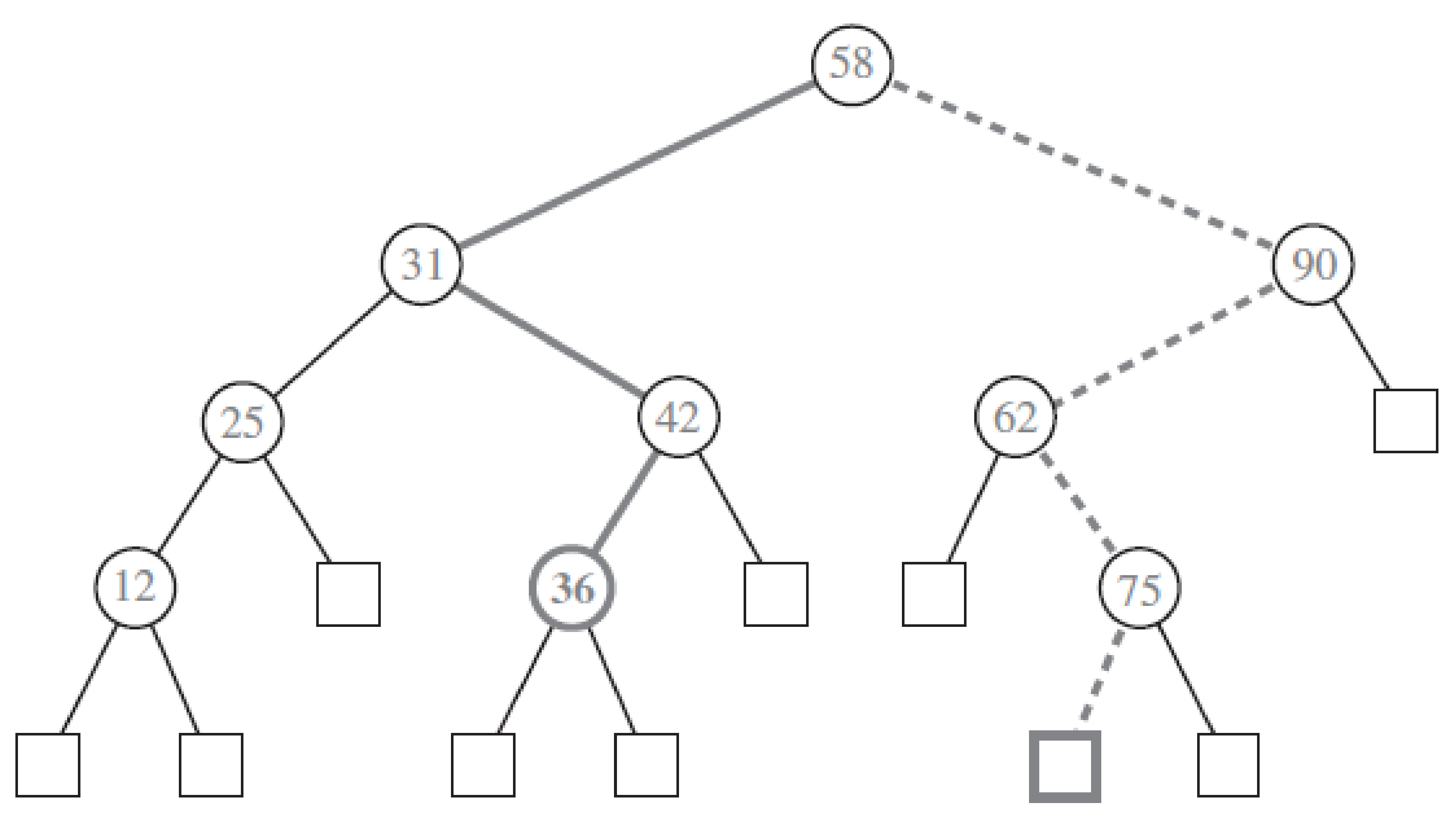
- رابطه بازگشتی:

$$\mathrm{T(n)}\le\begin{cases}\mathrm{b} & \text{if } \mathrm{n}<2\\ \mathrm{T}\left(\frac{\mathrm{n}}{2}\right)+\mathrm{b} & \text{else}\end{cases}\rightarrow \mathrm{O}(\log \mathrm{n})$$

$$T(n)=af\left(\frac{n}{b}\right)+\theta\left(n^k\log^p n\right)$$

1. $a>b^k: T(n)=\theta\left(n^{\log_b a}\right)$
2. $a=b^k$:
   a) $p>-1: T(n)=\theta\left(n^{\log_b a}\log^{p+1} n\right)$
   b) $p=-1: T(n)=\theta\left(n^{\log_b a}\log\log n\right)$
   c) $p<-1: T(n)=\theta\left(n^{\log_b a}\right)$
3. $a<b^k$:
   a) $p\ge 0: T(n)=\theta\left(n^k\log^p n\right)$
   b) $p<0: T(n)=\theta\left(n^k\right)$

درخت جستجوی دودویی

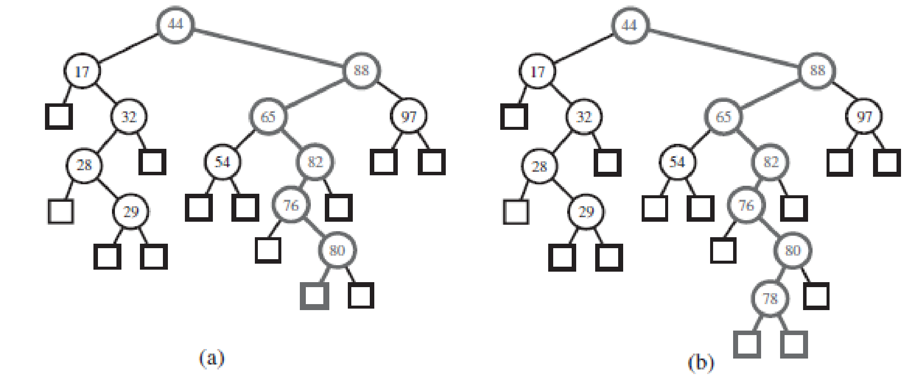


درخت جستجوی دودویی (ادامه)

وارد کردن عدد جدید

```
TREE-INSERT(T, z)
x = T.root                  // node being compared with z
y = NIL                     // y will be parent of z
while x ≠ NIL               // descend until reaching a leaf
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y                     // found the location—insert z with parent y
if y == NIL
    T.root = z              // tree T was empty
elseif z.key < y.key
    y.left = z
else y.right = z
```

نمونه وارد کردن عدد جدید



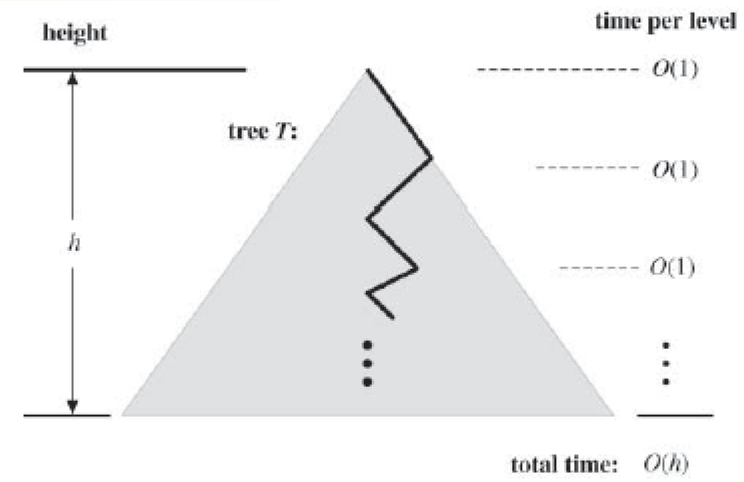
درخت جستجوی دودویی(ادامه)

جستجو

زمان اجرا(با شکل)

```
TREE-SEARCH(x, k)
if x == NIL or k == x.key
    return x
if k < x.key
    return TREE-SEARCH(x.left, k)
else return TREE-SEARCH(x.right, k)

ITERATIVE-TREE-SEARCH(x, k)
while x ≠ NIL and k ≠ x.key
    if k < x.key
        x = x.left
    else x = x.right
return x
```



درخت جستجوی دودویی (ادامه)

یافتن کمترین و بیشترین عنصر

```
TREE-MINIMUM(x)
while x.left ≠ NIL
    x = x.left
return x
```

```
TREE-MAXIMUM(x)
while x.right ≠ NIL
    x = x.right
return x
```

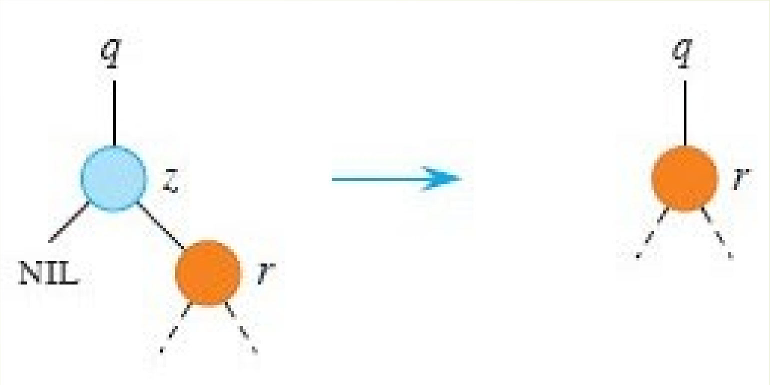
یافتن عنصر پسین و پیشین

```
TREE-SUCCESSOR(x)
if x.right ≠ NIL
    return TREE-MINIMUM(x.right)   // leftmost node in right subtree
else // find the lowest ancestor of x whose left child is an ancestor of x
    y = x.p
    while y ≠ NIL and x == y.right
        x = y
        y = y.p
    return y
```

```
function BST-PREDECESSOR(Node x)
    if x.left ≠ NIL then
        return BST-MAXIMUM(x.left)
    y ← x.p
    while y ≠ NIL and x = y.left do
        x ← y
        y ← y.p
    return y
```

درخت جستجوی دودویی (ادامه)

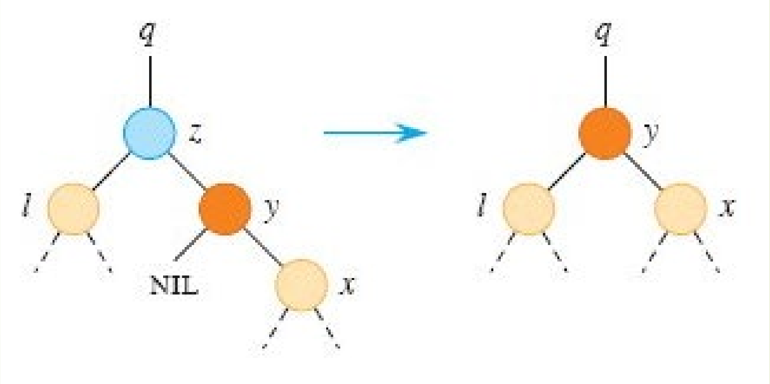
حذف حالت ۱



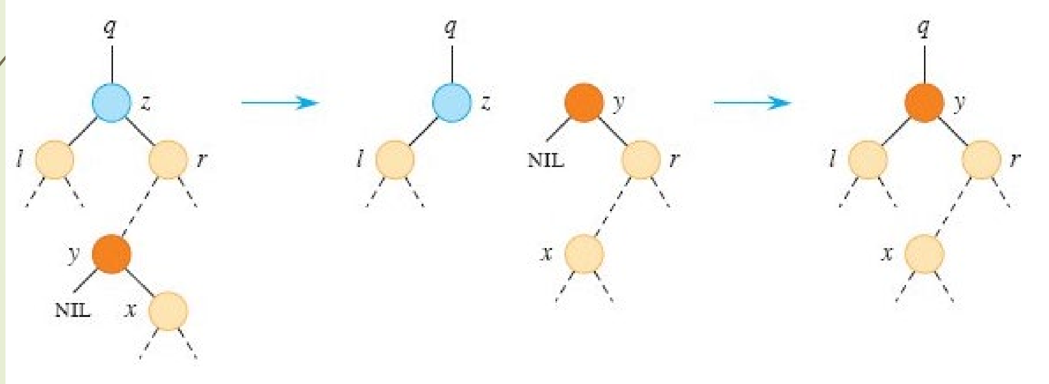
حذف حالت ۲

درخت جستجوی دودویی (ادامه)

حذف – حالت ۳ - الف



حذف – حالت ۳ - ب



درخت جستجوی دودویی (ادامه)

```
TRANSPLANT(T, u, v)
if u.p == NIL
    T.root = v
elseif u == u.p.left
    u.p.left = v
else u.p.right = v
if v ≠ NIL
    v.p = u.p
```

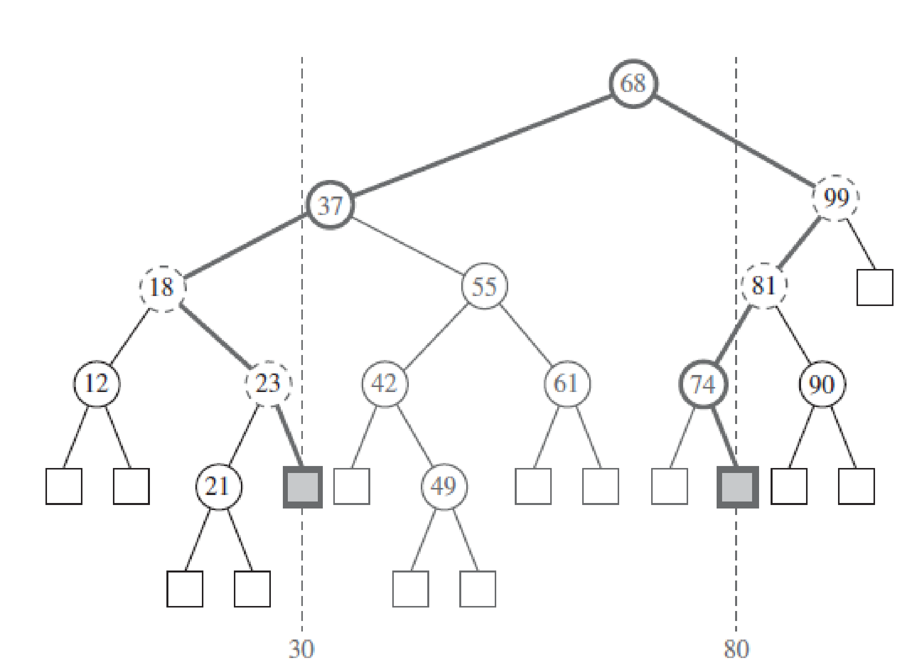
```
TREE-DELETE(T, z)
if z.left == NIL
    TRANSPLANT(T, z, z.right)          // replace z by its right child
elseif z.right == NIL
    TRANSPLANT(T, z, z.left)           // replace z by its left child
else y = TREE-MINIMUM(z.right)         // y is z's successor
    if y ≠ z.right                     // is y farther down the tree?
        TRANSPLANT(T, y, y.right)      // replace y by its right child
        y.right = z.right              // z's right child becomes
        y.right.p = y                  //     y's right child
    TRANSPLANT(T, z, y)                // replace z by its successor y
    y.left = z.left                    // and give z's left child to y,
    y.left.p = y                       //     which had no left child
```

درخت جستجوی دودویی (ادامه)

- قضیه: ما می‌توانیم عملیات‌های مجموعه پویا وارد کردن عنصر جدید، جستجو، حذف یک عنصر، یافتن مینیمم و ماکزیمم و یافتن successor و predecessor را پیاده‌سازی کنیم تا هر کدام در زمان O(h) روی درخت جستجوی دودویی با ارتفاع h اجرا شوند.
- سایت‌های شبیه‌سازی:
  - https://visualgo.net/en/bst
  - https://www.cs.usfca.edu/~galles/visualization/BST.html
  - https://yongdanielliang.github.io/animation/web/BST.html
  - https://www.mathwarehouse.com/programming/gifs/binary-search-tree.php

درخت جستجوی دودویی- جستجوی محدوده‌ها

نمونه



الگوریتم

Algorithm RangeQuery(k_1, k_2, v):

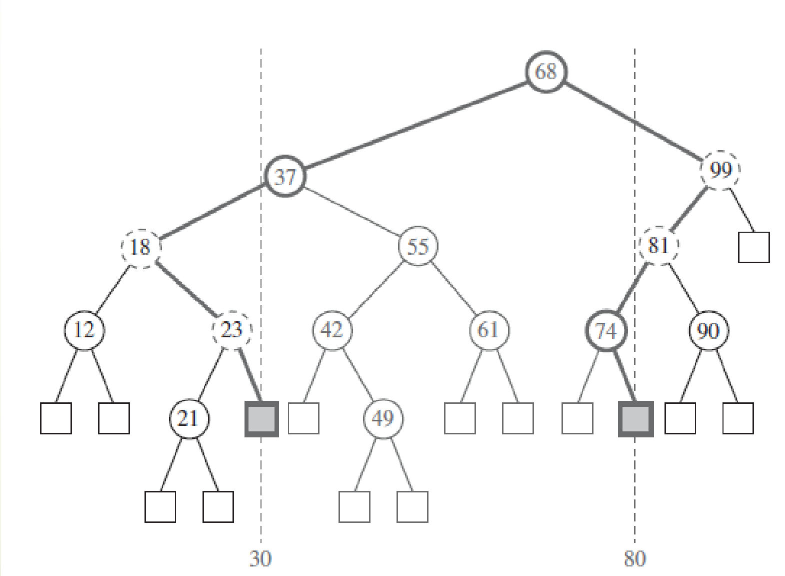
Input: Search keys k_1 and k_2, and a node v of a binary search tree T

Output: The elements stored in the subtree of T rooted at v whose keys are in the range $[k_1, k_2]$

if T.isExternal(v) **then**
  return $\emptyset$
if $k_1 \leq$ key$(v) \leq k_2$ **then**
  $L \leftarrow$ RangeQuery$(k_1, k_2, T.$leftChild$(v))$
  $R \leftarrow$ RangeQuery$(k_1, k_2, T.$rightChild$(v))$
  return $L \cup \{$element$(v)\} \cup R$
else if key$(v) < k_1$ **then**
  return RangeQuery$(k_1, k_2, T.$rightChild$(v))$
else if $k_2 <$ key(v) **then**
  return RangeQuery$(k_1, k_2, T.$leftChild$(v))$

درخت جستجوی دودویی- جستجوی محدوده‌ها (ادامه)

- فرض کنید P_1 مسیر جستجوی پیمایش شده هنگام انجام جستجو در درخت T برای کلید k_1 باشد. مسیر P_1 از ریشه T شروع می‌شود و به یک گره خارجی T ختم می‌شود. مسیر P_2 را به طور مشابه با توجه به k_2 تعریف کنید. ما هر گره v از T را به سه دسته تقسیم می‌کنیم:



- گره مرزی: گره v یک گره مرزی است اگر v متعلق به P_1 یا P_2 باشد. یک گره مرزی آیتمی را ذخیره می‌کند که کلید آن ممکن است داخل یا خارج از بازه $[k_1,k_2]$ باشد.
- گره درونی: گره v یک گره درونی است اگر v یک گره مرزی نباشد و v متعلق به یک زیردرخت باشد که در فرزند سمت راست یک گره P_1 یا در فرزند سمت چپ یک گره P_2 ریشه دارد. یک گره داخلی آیتمی را ذخیره می‌کند که کلید آن در بازه $[k_1,k_2]$ است.
- گره بیرونی: گره v یک گره بیرونی است اگر v یک گره مرزی نباشد و v متعلق به زیردرختی باشد که در فرزند چپ گره P_1 یا در فرزند راست گره P_2 ریشه دارد. یک گره خارجی آیتمی را ذخیره می‌کند که کلید آن خارج از بازه $[k_1,k_2]$ است.

درخت جستجوی دودویی- جستجوی محدوده‌ها (ادامه)

- قضیه: یک درخت جستجوی دودویی با ارتفاع h که n مورد را ذخیره می‌کند، عملیات جستجوی محدوده را با عملکرد زیر پشتیبانی می‌کند:

1. فضای مورد استفاده $O(n)$ است.
2. عملیات findAllInRange به زمان $O(h + s)$ نیاز دارد، جایی که s تعداد عناصر گزارش شده است.
3. عملیات درج و حذف هر عنصر $O(h)$ زمان می‌برد.

درخت جستجوی دودویی- جستجوی محدوده‌ها (ادامه)

- اثبات زمان اجرا:
  - ما هیچ گره خارجی را بازدید نمی‌کنیم.
  - ما حداکثر از گره‌های مرزی $2h + 1$ بازدید می‌کنیم، جایی که h ارتفاع درخت است، زیرا گره‌های مرزی در مسیرهای جستجوی P_1 و P_2 هستند و حداقل یک گره مشترک دارند (ریشه درخت).
  - هر بار که از یک گره داخلی v بازدید می‌کنیم، از کل زیردرخت T_v که ریشه در v دارد نیز بازدید می‌کنیم و تمام عناصر ذخیره شده در گره‌های داخلی T_v را به مجموعه گزارش شده اضافه می‌کنیم. اگر T_v آیتم‌های s_v را در خود جای دهد، $2s_v + 1$ گره دارد. گره‌های داخلی را می‌توان به j زیردرخت‌های جدا $T_1, T_2, \ldots, T_j$، تقسیم کرد که در فرزندان گره‌های مرزی ریشه دارد، جایی که $j \le 2h$. حال فرض کنید s_i نشان‌دهنده تعداد آیتم‌های ذخیره شده در درخت T_i باشد، تعداد کل گره‌های داخلی بازدید شده برابر است با:

$$\sum_{i=1}^{j}(2s_i + 1) = 2s + j \le 2s + 2h$$

  - بنابراین، حداکثر $2s + 4h + 1$ گره از T دیده می‌شود و عملیات findAllInRange در زمان $O(h + s)$ اجرا می‌شود.

ساخت درخت جستجوی دودویی به صورت تصادفی

- قضیه: ارتفاع مورد انتظار یک درخت جستجوی باینری تصادفی ساخته شده روی n کلید متمایز است از مرتبه $O(\log n)$ می‌باشد.
- پیش‌نیازهای اثبات:
  - خطی بودن امید ریاضی:

$$E[aX] = aE[X]$$

  - اگر X و Y دو متغیر تصادفی نامنفی باشند:

$$E(\max(X, Y)) \leq E[X] + E[Y]$$

  - یک رابطه ترکیبیاتی:

$$\sum_{i=0}^{n-1} \binom{i+3}{3} = \binom{n+3}{4}$$

  - نامساوی Jensen: اگر تابع f محدب باشد، برای هر توزیع P روی M داریم:

$$E_{m\sim P}[f(X(m))] \geq f(E_{m\sim P}[X(m)])$$

ساخت درخت جستجوی دودویی به صورت تصادفی (ادامه)

- اثبات $E(\max(X,Y)) \leq E[X] + E[Y]$:

$$X + Y = \max(X, Y) + \min(X, Y) \Rightarrow E[X + Y] = E[\max(X, Y) + \min(X, Y)]$$

در ادامه با استفاده از خاصیت خطی بودن امید ریاضی داریم:

$$E[X] + E[Y] = E[\max(X, Y)] + E[\min(X, Y)] \Rightarrow E[\max(X, Y)] = E[X] + E[Y] - E[\min(X, Y)]$$

بنابراین از رابطه فوق می‌توان نتیجه گرفت:

$$E[\max(X, Y)] \leq E[X] + E[Y]$$

ساخت درخت جستجوی دودویی به صورت تصادفی (ادامه)

- اثبات رابطه ترکیبیاتی:

$$\sum_{i=0}^{n-1}\binom{i+3}{3}=\binom{n+3}{4}$$

استفاده از استقرا:

✓ پایه استقرا:

$$n=1: \sum_{i=0}^{0}\binom{i+3}{3}=\binom{4}{4}\Rightarrow\binom{3}{3}$$

$$=\binom{4}{4}=1$$

✓ فرض استقرا:

$$n=k: \sum_{i=0}^{k-1}\binom{i+3}{3}=\binom{k+3}{4}$$

✓ حکم استقرا:

$$n=k+1: \sum_{i=0}^{k}\binom{i+3}{3}=\binom{k+4}{4}$$

$$\sum_{i=0}^{k}\binom{i+3}{3}=\binom{k+3}{3}+\sum_{i=0}^{k-1}\binom{i+3}{3}$$

$$\Rightarrow \frac{(k+3)!}{3!\,k!}+\binom{k+3}{4}$$

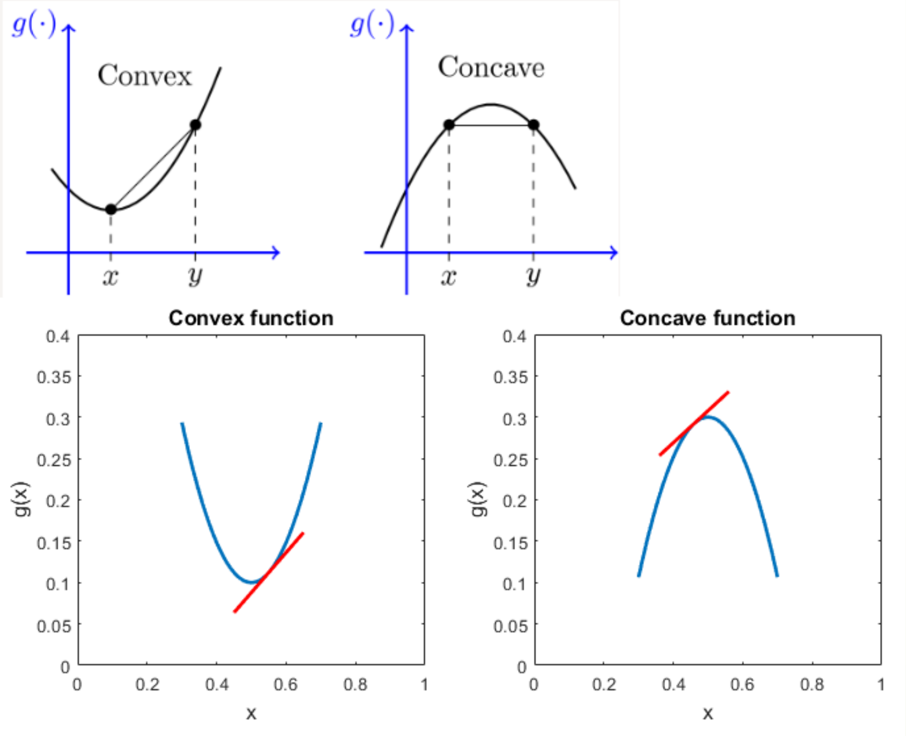
$$=\frac{(k+3)!}{3!\,k!}+\frac{(k+3)!}{4!\,(k-1)!}$$

$$=\frac{4\times(k+3)!+k\times(k+3)!}{4!\,k!}$$

$$=\frac{(k+4)\times(k+3)!}{4!\,k!}$$

$$=\frac{(k+4)!}{4!\,(k+4-4)!}=\binom{k+4}{4}\ \blacksquare$$

ساخت درخت جستجوی دودویی به صورت تصادفی (ادامه)



- فرض کنید $g: I \to \mathbb{R}$ و $\alpha \in [0,1]$
- تعریف محدب(Convex) بودن:

$$g(\alpha x + (1-\alpha)y) \leq \alpha g(x) + (1-\alpha) g(y)$$
$$g''(x) \geq 0$$

- تعریف مقعر(Concave) بودن:

$$g(\alpha x + (1-\alpha)y) \geq \alpha g(x) + (1-\alpha) g(y)$$
$$g''(x) \leq 0$$

ساخت درخت جستجوی دودویی به صورت تصادفی (ادامه)

- اثبات نامساوی Jensen:

$$E_{m\sim P}[f(X(m))] = \sum_{i=1}^{\infty} p_i f(x_i) = p_1 f(x_1) + p_2 f(x_2) + \sum_{i=3}^{\infty} p_i f(x_i)$$

$$= (p_1 + p_2)\left[\frac{p_1}{p_1 + p_2} f(x_1) + \frac{p_2}{p_1 + p_2} f(x_2)\right] + \sum_{i=3}^{\infty} p_i f(x_i)$$

در ادامه با استفاده از خاصیت محدب بودن تابع f می‌توانیم بنویسیم:

$$E_{m\sim P}[f(X(m))] = (p_1 + p_2)\left[\frac{p_1}{p_1 + p_2} f(x_1) + \frac{p_2}{p_1 + p_2} f(x_2)\right] + \sum_{i=3}^{\infty} p_i f(x_i)$$

$$\geq (p_1 + p_2) f\left(\frac{p_1}{p_1 + p_2} x_1 + \frac{p_2}{p_1 + p_2} x_2\right) + \sum_{i=3}^{\infty} p_i f(x_i)$$

$$\geq f\left((p_1 + p_2)\left(\frac{p_1}{p_1 + p_2} x_1 + \frac{p_2}{p_1 + p_2} x_2\right) + \sum_{i=3}^{\infty} p_i x_i\right) = f\left(\sum_{i=1}^{\infty} p_i x_i\right) = f(E_{m\sim P}[X(m)])$$

ساخت درخت جستجوی دودویی به صورت تصادفی (ادامه)

- برای اثبات قضیه ابتدا مجموعه و متغیرهای تصادفی زیر را تعریف می‌کنیم:
  - ✓ مجموعه S شامل n کلید متمایز از 1 تا n:

$$S = \{1,2,\dots,n\} = \{x|x \in \mathbb{N} \ \wedge \ x \leq n\}$$

  - ✓ متغیر تصادفی X_n ارتفاع درخت دودویی ساخته شده به صورت تصادفی
  - ✓ متغیر تصادفی $Y_n = 2^{X_n}$ نشان‌دهنده ارتفاع نمایی
  - ✓ متغیر تصادفی R_n: نشان‌دهنده(ذخیره‌کننده) رتبه کلید انتخاب‌شده به عنوان ریشه
  - ✓ متغیر تصادفی $Z_{n,i}$:

$$Z_{n,i} = I\{R_n = i\}$$

- حال فرض کنید $R_n = i$ به عنوان ریشه انتخاب شود:

$$\Pr(R_n = i) = \frac{1}{n}, E[Z_{n,i}] = \frac{1}{n}$$

$$Y_n = 2\max(Y_{i-1}, Y_{n-i})$$

ساخت درخت جستجوی دودویی به صورت تصادفی (ادامه)

- در ادامه می‌توان نوشت:

$$Y_n = \sum_{i=1}^{n} Z_{n,i} 2\max(Y_{i-1}, Y_{n-i}) \Rightarrow E[Y_n] = E\left[\sum_{i=1}^{n} Z_{n,i} 2\max(Y_{i-1}, Y_{n-i})\right]$$

$$= \sum_{i=1}^{n} E\left[Z_{n,i} 2\max(Y_{i-1}, Y_{n-i})\right] = \sum_{i=1}^{n} E\left[Z_{n,i}\right] E[2\max(Y_{i-1}, Y_{n-i})]$$

$$= \sum_{i=1}^{n} \frac{1}{n} \times 2E[\max(Y_{i-1}, Y_{n-i})] = \frac{2}{n}\sum_{i=1}^{n} E[\max(Y_{i-1}, Y_{n-i})] \le \frac{2}{n}\sum_{i=1}^{n} E[Y_{i-1}] + E[Y_{n-i}]$$

$$= \frac{4}{n}\sum_{i=1}^{n-1} E[Y_i] \le \frac{4}{n}\sum_{i=1}^{n-1} \frac{1}{4}\binom{i+3}{3} = \frac{1}{n}\sum_{i=1}^{n-1}\binom{i+3}{3} = \frac{1}{n}\binom{n+3}{4} = \frac{1}{4}\binom{n+3}{3}$$

ساخت درخت جستجوی دودویی به صورت تصادفی (ادامه)

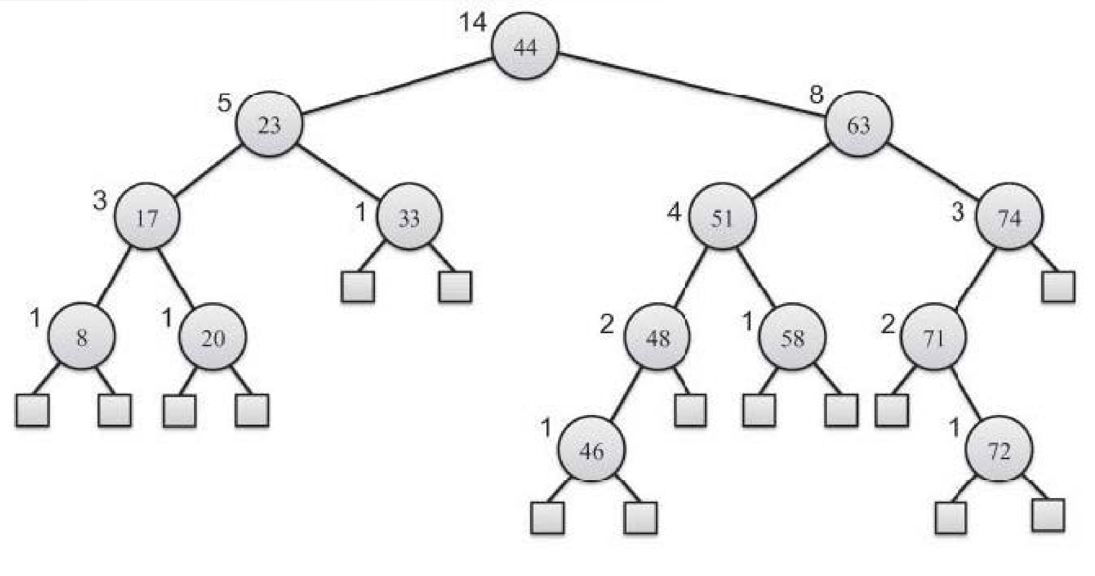
- در ادامه با استفاده نامساوی Jensen داریم:

$$2^{E[X_n]} \leq E[2^{X_n}] = E[Y_n] \Rightarrow 2^{E[X_n]} \leq \frac{1}{4}\binom{n+3}{3} = \frac{1}{4} \times \frac{(n+3)(n+2)(n+1)}{6}$$
$$= \frac{(n+3)(n+2)(n+1)}{24}$$

با گرفتن لگاریتم از دو طرف نامساوی داریم:

$$\log 2^{E[X_n]} \leq \log \frac{(n+3)(n+2)(n+1)}{24} \Rightarrow E[X_n] \leq \log(n+3)(n+2)(n+1) - \log 24$$
$$= \log(n+3) + \log(n+2) + \log(n+1) - \log 24 \Rightarrow E[X_n] = O(\log n)$$

درخت جستجوی دودویی- جستجوی بر مبنای شاخص



- فرض کنید عملیات زیر بر روی درخت جستجوی دودویی ساخته‌شده T انجام می‌پذیرد:

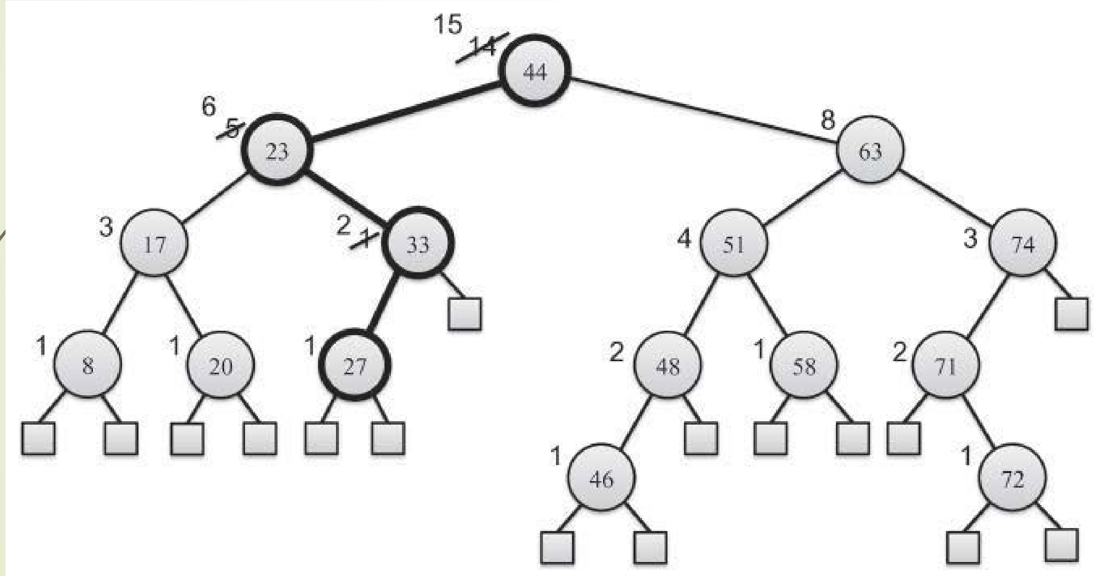
$$\forall 1 \le i \le n$$

select(i): Return the item with ith smallest key

- ایده: افزودن یک مشخصه به دیگر به هر گره علاوه‌بر زوج مرتب‌های کلید-مقدار(key-value)

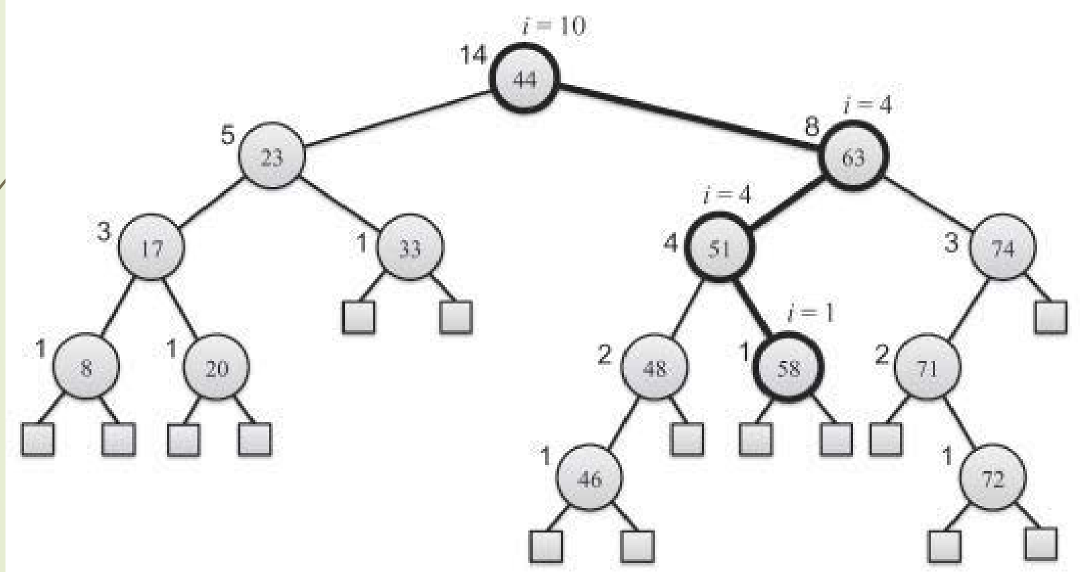
به هر گره v یک مشخصه n_v اضافه می‌کنیم که نشان‌دهنده تعداد آیتم‌های ذخیره‌شده در زیر درخت T با ریشه v است.

درخت جستجوی دودویی- جستجوی مبتنی بر شاخص(ادامه)



- بروزرسانی‌ها روی درخت جستجوی دودویی افزوده:
  - وارد کردن عنصر جدید
  - حذف عنصر

درخت جستجوی دودویی- جستجوی مبتنی بر شاخص(ادامه)



Algorithm TreeSelect(i, v, T):

Input: Search index i and a node v of a binary search tree T

Output: The item with ith smallest key stored in the subtree of T rooted at v

Let $w \leftarrow T$.leftChild(v)
if $i \le n_w$ **then**
  return TreeSelect(i, w, T)
else if $i = n_w + 1$ **then**
  return (key(v), element(v))
else
  return TreeSelect$(i - n_w - 1, T$.rightChild$(v), T)$

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14
value	8	17	20	23	33	44	46	48	51	58	63	71	72	74

سؤال؟

مثال‌های حل شده

مثال ۱ (سؤال ۲-۱۲.۲ صفحه ۲۹۳ کتاب CLRS 3rd edition)

Algorithm TREE-MINIMUM(x)

```
if x.left ≠ NIL then
    return TREE − MINIMUM(x.left)
else
    return x
end if
```

Algorithm TREE-MAXIMUM(x)

```
if x.right ≠ NIL then
    return TREE − MAXIMUM(x.right)
else
    return x
end if
```

- نسخه‌های بازگشتی
  - TREE-MINIMUM
  - TREE-MAXIMUM

را بنویسید.

12.2-2
Write recursive versions of TREE-MINIMUM and TREE-MAXIMUM.

مثال ۲ (سؤال ۱۲.۴-۴ صفحه ۳۰۳ کتاب CLRS 3rd edition)

- نشان دهید تابع $f(x) = 2^x$ محدب است.

12.4-4
Show that the function $f(x) = 2^x$ is convex.

- برای محدب بودن تابع باید طبق تعاریف نشان دهیم که مشتق دوم تابع بزرگتر-مساوی صفر می‌باشد، یعنی باید نشان دهیم: $f''(x) \geq 0$

$$f(x) = 2^x \Rightarrow \ln f(x) = \ln 2^x \Rightarrow \ln f(x) = x \ln 2$$

در ادامه از دو طرف تساوی مشتق می‌گیریم:

$$(\ln f(x))' = (x \ln 2)' \Rightarrow \frac{f'(x)}{f(x)} = \ln 2 \Rightarrow f'(x) = f(x) \times \ln 2 = 2^x \ln 2$$

در ادامه برای مشتق دوم همانند مشتق مرتبه اول دوباره از دو طرف تساوی لگاریتم طبیعی(لگاریتم در مبنای عدد نپر(e)) می‌گیریم و سپس از دو طرف تساوی مشتق می‌گیریم، یعنی داریم:

$$f'(x) = 2^x \ln 2 \Rightarrow \ln f'(x) = \ln(2^x \ln 2) \Rightarrow \ln f'(x) = x \ln 2 + \ln \ln 2 \Rightarrow (\ln f'(x))' = (x \ln 2 + \ln \ln 2)' \Rightarrow \frac{f''(x)}{f'(x)} = \ln 2 \Rightarrow f''(x) = f'(x) \times \ln 2 = 2^x (\ln 2)^2$$

عبارت نهایی حاصلضرب یک عبارت نمایی در یک عدد مثبت است و بنابراین همیشه مثبت است و بنابراین $f''(x) \geq 0$ و به این ترتیب ثابت می‌شود تابع $f(x)$ محدب است.

مثال ۳ (سؤال ۴-۱۲ صفحه ۳۰۶ کتاب CLRS 3rd edition – تعداد درخت‌های دودویی متفاوت)

- اجازه دهید b_n تعداد درختان دودویی مختلف را با n گره نشان دهد. در این مسئله، فرمولی برای b_n و همچنین تخمین مجانبی پیدا خواهید کرد.

a. نشان دهید اگر $b_0 = 1$ باشد، آنگاه به ازای $n \geq 1$ داریم:

$$b_n = \sum_{k=0}^{n-1} b_k b_{n-k-1}$$

- برای بدست آوردن رابطه بازگشتی مورد نظر کافی است همانند اثبات قضیه در نظر بگیریم مقدار i به عنوان ریشه انتخاب شده باشد بنابراین زیردرخت سمت چپ با مقادیر 1 تا $i-1$ ساخته می‌شود و زیردرخت سمت راست با مقادیر $i+1$ تا n ساخته می‌شود. تعداد گره‌هایی که در زیر درخت سمت چپ وجود دارند $i-1$ و تعداد گره‌هایی که در زیردرخت سمت چپ وجود دارند $n-i$ هستند و بنابراین تعداد زیردرخت‌های سمت چپ و راستی که با توجه به تعداد گره‌هایی در دسترس مطابق نمادگذاری می‌توان ساخت عبارتند از: $b_{i-1} \times b_{n-i}$ و در ادامه باید در نظر داشته باشیم که ریشه i هر یک از مقادیر 1 تا n می‌تواند باشد، بنابراین رابطه بازگشتی نهایی برای b_n برابر است با:

$$b_n = \sum_{i=1}^{n} b_{i-1} b_{n-i} \overset{i-1=k}{\widehat{=}} \sum_{k=0}^{n-1} b_k b_{n-k-1}$$

مثال ۴ (سؤال۴-۱۲ صفحه ۳۰۶ کتاب CLRS 3rd edition – تعداد درخت‌های دودویی متفاوت)

- فرض کنید $B(x)$ یک تابع مولد به صورت زیر باشد:

$$B(x) = \sum_{n=0}^{\infty} b_n x^n$$

باشد. با استفاده از رابطه مثال ۳ ثابت کنید:

$$B(x) = 1 + x\big(B(x)\big)^2$$

و نشان دهید فرم بسته آن به صورت زیر است:

$$B(x) = \frac{1}{2x}\left(1 - \sqrt{1-4x}\right)$$

$$\begin{aligned}
B(x) &= \sum_{n=0}^{\infty} b_n x^n = 1 + \sum_{n=1}^{\infty} b_n x^n \\
&= 1 + \sum_{n=1}^{\infty}\sum_{k=0}^{n-1} b_k b_{n-k-1}\, x^n \\
&= 1 + x\sum_{n=1}^{\infty}\sum_{k=0}^{n-1} b_k x^k b_{n-k-1}\, x^{n-k-1} \\
&= 1 + x\sum_{n=0}^{\infty}\sum_{k=0}^{n} b_k x^k b_{n-k}\, x^{n-k} \\
&= 1 + x\big(B(x)\big)^2
\end{aligned}$$

حاصلضرب کوشی دو سری توانی:

$$\left(\sum_{i=0}^{\infty} a_i x^i\right).\left(\sum_{j=0}^{\infty} a_j x^j\right) = \left(\sum_{k=0}^{\infty} c_k x^k\right) \text{ where } c_k = \sum_{l=0}^{k} a_l b_{k-l}$$

مثال ۴ (سؤال ۴-۱۲ صفحه ۳۰۶ کتاب CLRS 3rd edition – تعداد درخت‌های دودویی متفاوت)(ادامه)

- برای فرم بسته داریم:

$$B(x) = 1 + x\big(B(x)\big)^2 \Rightarrow x\big(B(x)\big)^2 - B(x) - 1 = 0 \Rightarrow B(x) = \frac{1 \pm \sqrt{1-4x}}{2x}$$

از طرفی داریم $B(0) = 0$ و با توجه به اینکه $x = 0$ در دامنه $B(x)$ نمی‌باشد، آنگاه $B(0)$ را در حالت حدی بررسی می‌نماییم و می‌نویسیم:

$$\lim_{x \to 0} \frac{1 + \sqrt{1-4x}}{2x} = \frac{2}{0}$$

حاصل این حد تعریف نشده می‌باشد و بنابراین این جواب غیرممکن است.

$$\lim_{x \to 0} \frac{1 - \sqrt{1-4x}}{2x} = \frac{0}{0} \overset{\text{رفع ابهام}}{\cong} \frac{1}{1} = 1$$

بنابراین فرم بسته به صورت زیر خواهد بود:

$$B(x) = \frac{1 - \sqrt{1-4x}}{2x}$$