طراحی و تحلیل الگوریتم‌ها

مبحث شانزدهم

مباحث پیشرفته در ساختمان داده‌ها

هیپ‌های فیبوناچی

Fibonacci Heaps

کاظم فولادی
دانشکده مهندسی برق و کامپیوتر
دانشگاه تهران

http://courses.fouladi.ir/algorithm

هیپ فیبوناچی

FIBONACCI HEAP

زمان اجرای عملیات در هیپ فیبوناچی مشابه هیپ دوجمله‌ای است،
با این تفاوت که:
عملیاتی که شامل حذف یک عنصر نیستند، در زمان سرشکن شده‌ی $O(1)$ اجرا می‌شوند.

ساختار هیپ فیبوناچی نسبت به هیپ دوجمله‌ای ساده‌تر است.

(کاری که ساختار هیپ را حفظ می‌کند، می‌تواند تا زمانی که اجرای آن آسان شود به تأخیر انداخته شود.)

هیپ فیبوناچی

FIBONACCI HEAP

هیپ فیبوناچی
Fibonacci Heaps

مشابه هیپ دوجمله‌ای با این تفاوت که
لیست درخت‌های داخل هیپ دوجمله‌ای مرتب شده است، اما
لیست درخت‌های داخل هیپ فیبوناچی نامرتب است.

❈ در لیست ریشه‌ها لزوماً یک درخت از مرتبه‌ی B_k وجود ندارد (امکان وجود درخت‌های هم‌مرتبه).

هیپ فیبوناچی

بازنمایی

FIBONACCI HEAP

ساختار گره:

$\text{parent}[x]$	
$\text{key}[x]$	
$\text{degree}[x]$	
$\text{Mark}[x]$	
$\text{left}[x]$	$\text{right}[x]$
$\text{child}[x]$	

- $\text{parent}[x]$: اشاره‌گری به والد گره (در ریشه NIL)
- $\text{key}[x]$: مقدار کلید گره
- $\text{degree}[x]$: تعداد فرزندان گره
- $\text{Mark}[x]$: **فلید علامت** : «مقدار بولی» آیا گره x از آخرین باری که فرزند گره‌ی دیگری شده است، تغییر کرده است؟
- $\text{left}[x]$: اشاره‌گری به چپ‌ترین همزاد گره — در صورت عدم وجود به خودش اشاره می‌کند.
- $\text{right}[x]$: اشاره‌گری به راست‌ترین همزاد گره
- $\text{child}[x]$: اشاره‌گری به یکی از فرزندان گره

ساختار ساختمان داده:

از لیست پیوندی دوطرفه‌ی حلقوی استفاده می‌شود:
(مزیت حذف یک گره در $O(1)$ و الحاق دو لیست در زمان $O(1)$)

اشاره‌گر $\min[H]$ به ریشه‌ی درخت دارای کلید می‌نیمم (در لیست ریشه)

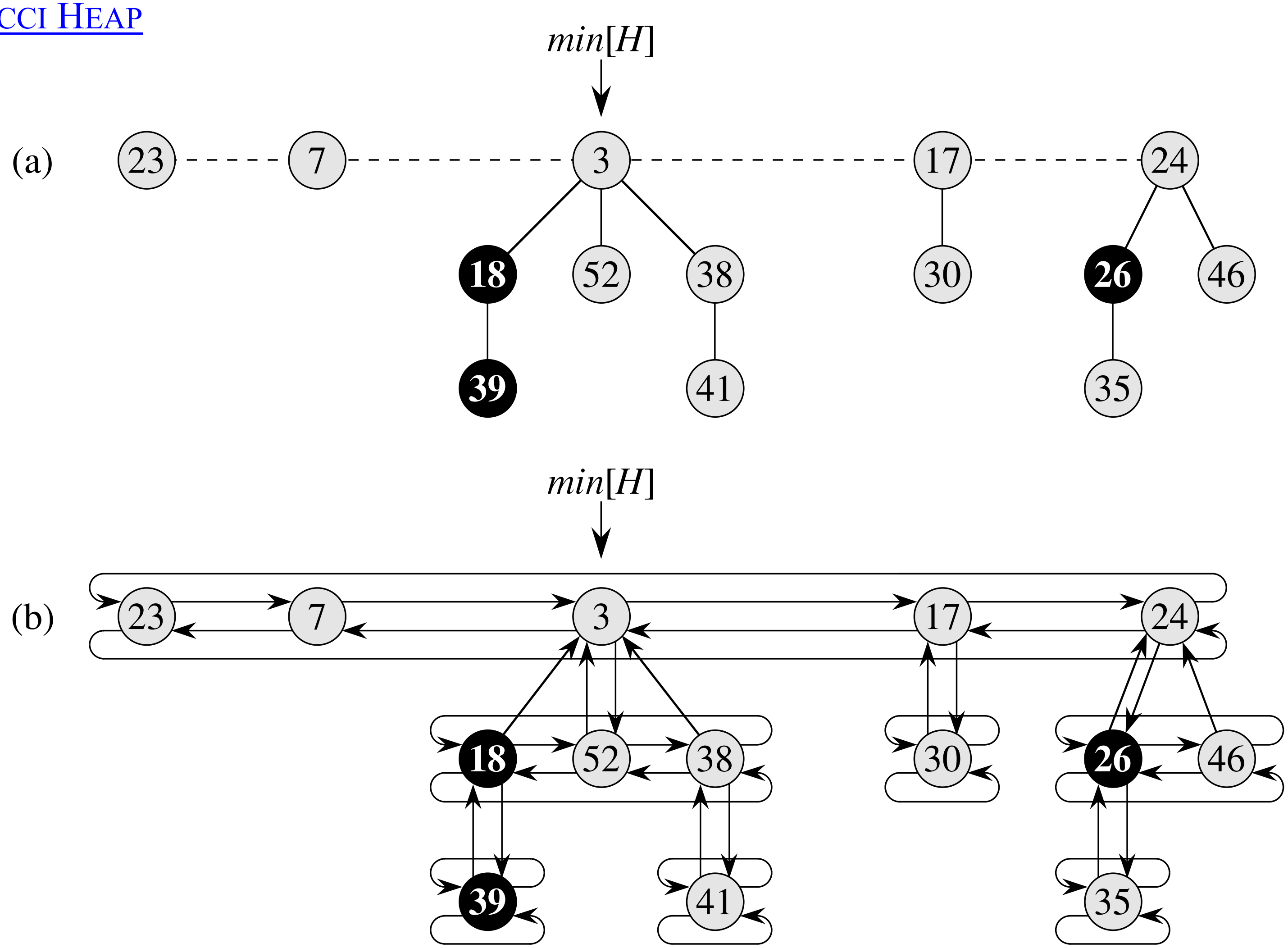
$\min[H] = \text{NIL}$ به معنی خالی بودن هیپ است.

$n(H)$ = تعداد گره‌های جاری در H

هیپ فیبوناچی

مثال

FIBONACCI HEAP



هیپ فیبوناچی

تابع پتانسیل برای تحلیل سرشکنی کارآیی عملیات هیپ فیبوناچی

FIBONACCI HEAP

$$\Phi(H) = t(H) + 2m(H)$$

تعداد درخت‌ها در لیست ریشه‌ی هیپ

تعداد گره‌های علامت‌دار در هیپ

پتانسیل مجموعه‌ای از هیپ‌های فیبوناچی برابر با جمع پتانسیل‌های هیپ‌های فیبوناچی تشکیل‌دهنده‌ی آن است.

درخت دوجمله‌ای نامرتب

UNORDERED BINOMIAL TREE

درخت دوجمله‌ای نامرتب
Onordered Binomial Tree

یک درخت نامرتب U_k (که موقعیت قرارگیری فرزندان هر گره در آن مهم نیست) به صورت بازگشتی تعریف می‌شود:

- اگر $k = 0$ باشد، یک گره‌ی تنها داریم.
- اگر $k > 0$ باشد، U_k به صورت زیر ساخته می‌شود:

«ریشه‌ی یکی از درخت‌های U_{k-1} را به عنوان فرزند ریشه‌ی درخت U_{k-1} دیگر اضافه می‌کنیم.»

درخت دوجمله‌ای نامرتب

ویژگی‌ها

Unordered Binomial Tree

ویژگی‌ها مشابه درخت دوجمله‌ای مرتب است، با یک تفاوت:

درخت دوجمله‌ای مرتب

درجه‌ی ریشه‌ی درخت B_k برابر با k است؛
که بزرگ‌تر از درجه‌ی هر گره‌ی دیگر است.
فرزندان ریشه، ریشه‌های زیردرخت‌های B_0، B_1، …، B_{k-1} از راست به چپ هستند.

درخت دوجمله‌ای نامرتب

درجه‌ی ریشه‌ی درخت U_k برابر با k است؛
که بزرگ‌تر از درجه‌ی هر گره‌ی دیگر است.
فرزندان ریشه، ریشه‌های زیردرخت‌های U_0، U_1، …، U_{k-1} به هر ترتیبی هستند.

هیپ فیبوناچی

مجموعه‌ای از درخت‌های دودویی نامرتب

FIBONACCI HEAP

هر هیپ فیبوناچی، مجموعه‌ای از درخت‌های دوجمله‌ای نامرتب است.

برای تحلیل‌های سرشکنی فرض می‌کنیم
کران بالای درجه‌ی همه‌ی گره‌ها در هیپ فیبوناچی با n گره، معلوم است:

$$D(n) \leq \lfloor \lg n \rfloor$$

Thus, if an n-node Fibonacci heap is a collection of unordered binomial trees, then $D(n) = \lg n$.

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

MERGEABLE-HEAP OPERATIONS

MAKE-FIB-HEAP()
FIB-HEAP-INSERT(H,x)
FIB-HEAP-MINIMUM(H)
FIB-HEAP-UNION(H_1,H_2)
FIB-HEAP-EXTRACT-MIN(H)

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

ایجاد یک هیپ جدید

MAKE-FIB-HEAP()
برای شیئ H حافظه می‌گیرد و اشاره‌گر به آن را برمی‌گرداند که $\min[H] = \text{NIL}$

$$\Phi(H) = t(H) + 2m(H)$$

$$t(H) = m(H) = 0 \Rightarrow \Phi(H) = 0$$

$$\Downarrow$$

$$\hat{c}_i = c_i = O(1)$$

هزینه‌ی سرشکن‌شده

هزینه‌ی واقعی

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

درج یک گره

FIB-HEAP-INSERT(H,x)

گره‌ی جدیدی ساخته می‌شود و به لیست ریشه‌ی H اضافه می‌شود.

```
FIB-HEAP-INSERT(H, x)
degree[x] ← 0
p[x] ← NIL
child[x] ← NIL
left[x] ← x
right[x] ← x
mark[x] ← FALSE
concatenate the root list containing x with root list H
if min[H] = NIL or key[x] < key[min[H]]
   then min[H] ← x
n[H] ← n[H] + 1
```

روال درج برای ترکیب درخت‌های داخل هیپ فیبوناچی تلاشی نمی‌کند:

اگر k عمل پی‌درپی درج رخ دهد، k درخت تک گره‌ای به لیست ریشه اضافه می‌شود.

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

درج یک گره: مثال

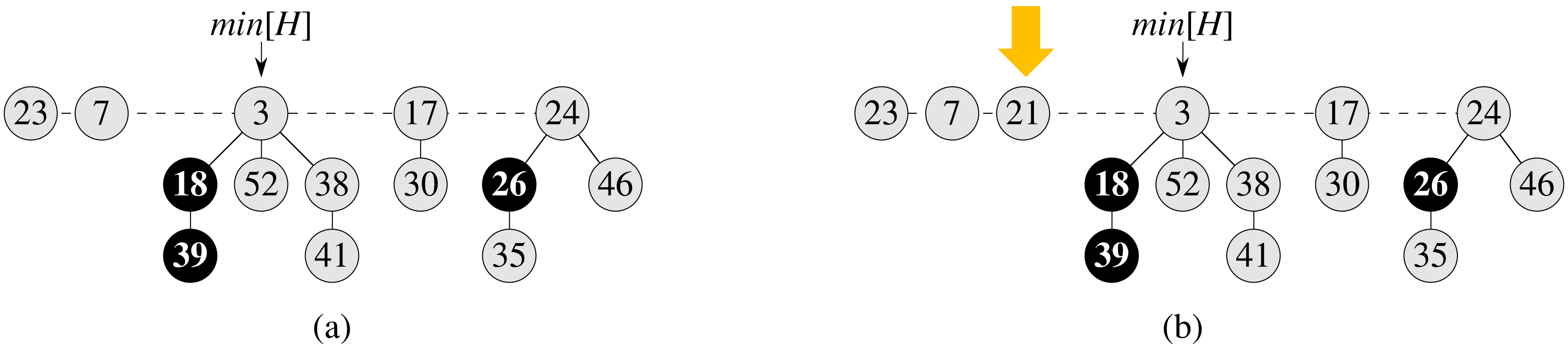


Figure 20.2 Inserting a node into a Fibonacci heap. **(a)** A Fibonacci heap H. **(b)** Fibonacci heap H after the node with key 21 has been inserted. The node becomes its own min-heap-ordered tree and is then added to the root list, becoming the left sibling of the root.

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

درج یک گره: زمان اجرای سرشکن‌شده

$$\Phi(H) = t(H) + 2m(H)$$

$$\begin{aligned}\Delta\Phi &= \Phi(H') - \Phi(H) \\ &= [t(H) + 1 + 2m(H)] - [t(H) + 2m(H)] \\ &= 1\end{aligned}$$

$$\Downarrow$$

$$\hat{c}_i = c_i + \Delta\Phi = O(1) + 1 = O(1)$$

هزینه‌ی سرشکن‌شده

هزینه‌ی واقعی

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

یافتن گره‌ی می‌نیمم

$\textsc{Fib-Heap-Minimum}(H)$
کوچک‌ترین عنصر هیپ فیبوناچی را برمی‌گرداند.

این گره با اشاره‌گر $\min[H]$ مستقیماً قابل دست‌یابی است.

$$\Phi(H) = t(H) + 2m(H)$$
$$\Delta\Phi(H) = 0$$
$$\Downarrow$$
$$\hat{c}_i = c_i = O(1)$$

هزینه‌ی سرشکن‌شده

هزینه‌ی واقعی

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

اجتماع دو هیپ فیبوناچی

FIB-HEAP-UNION(H_1, H_2)

دو هیپ فیبوناچی را با هم اجتماع می‌گیرد.

FIB-HEAP-UNION(H_1, H_2)

$H \leftarrow$ MAKE-FIB-HEAP()
$min[H] \leftarrow min[H_1]$
concatenate the root list of H_2 with the root list of H
if $(min[H_1] = \text{NIL})$ or $(min[H_2] \neq \text{NIL}$ and $key[min[H_2]] < key[min[H_1]])$
    then $min[H] \leftarrow min[H_2]$
$n[H] \leftarrow n[H_1] + n[H_2]$
free the objects H_1 and H_2
return H

این روال به‌سادگی لیست ریشه‌های دو هیپ را به هم متصل می‌کند
و سپس min نهایی را بین min آنها تعیین می‌کند.

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

اجتماع دو هیپ فیبوناچی: زمان اجرای سرشکن‌شده

$$\Phi(H) = t(H) + 2m(H)$$

$$\begin{aligned}\Delta\Phi &= \Phi(H) - (\Phi(H_1) + \Phi(H_2)) \\ &= [t(H) + 2m(H)] - \big([t(H_1) + 2m(H_1)] + [t(H_2) + 2m(H_2)]\big) \\ &= 0 \\ &\quad (\text{since, } \ m(H) = m(H_1) + m(H_2), \ t(H) = t(H_1) + t(H_2))\end{aligned}$$

$$\Downarrow$$

$$\hat{c}_i = c_i + \Delta\Phi = O(1) + 0 = O(1)$$

هزینه‌ی سرشکن‌شده

هزینه‌ی واقعی

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

استخراج گره‌ی می‌نیمم

FIB-HEAP-EXTRACT-MIN(H)

ریشه‌ی دارای کلید می‌نیمم را حذف می‌کند و مقدار آن را برمی‌گرداند.

```
FIB-HEAP-EXTRACT-MIN(H)
z ← min[H]
if z ≠ NIL
   then for each child x of z
            do add x to the root list of H
               p[x] ← NIL
        remove z from the root list of H
        if z = right[z]
           then min[H] ← NIL
           else min[H] ← right[z]
                CONSOLIDATE(H)
        n[H] ← n[H] − 1
return z
```

این روال، همه‌ی فرزندان گره‌ی می‌نیمم (z) را به لیست ریشه اضافه می‌کند و z را از لیست ریشه حذف می‌کند.
اگر z همزادی نداشته باشد، $min \leftarrow \text{NIL}$.
وگرنه min را برابر با همزاد راست z قرار می‌دهد
و روال CONSOLIDATE را فراخوانی می‌کند تا گره‌های داخل لیست ریشه دارای درجه‌های متمایز شوند.

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

استخراج گره‌ی می‌نیمم: عمل اتحاد

```
CONSOLIDATE(H)
for i ← 0 to D(n[H])
     do A[i] ← NIL
for each node w in the root list of H
     do x ← w
        d ← degree[x]
        while A[d] ≠ NIL
            do y ← A[d]          ▷ Another node with the same degree as x.
               if key[x] > key[y]
                 then exchange x ↔ y
               FIB-HEAP-LINK(H, y, x)
               A[d] ← NIL
               d ← d + 1
        A[d] ← x
min[H] ← NIL
for i ← 0 to D(n[H])
     do if A[i] ≠ NIL
          then add A[i] to the root list of H
               if min[H] = NIL or key[A[i]] < key[min[H]]
                 then min[H] ← A[i]
```

روال CONSOLIDATE

تا زمانی که گره‌های داخل لیست ریشه دارای درجه‌های متمایز شوند:

- یافتن ریشه‌های x و y در لیست ریشه با درجه‌ی یکسان که $\text{key}[x] \leq \text{key}[y]$
- اتصال y به x ؛
  حذف y از لیست ریشه ؛
  قراردادن y به‌عنوان فرزند x
  (با روال FIB-HEAP-LINK)

استفاده از آرایه‌ی کمکی:
$A[0\ ..\ D(n(H))]$
اگر $A[i] = y$ یعنی
y یک ریشه با درجه‌ی i است.

روال FIB-HEAP-LINK

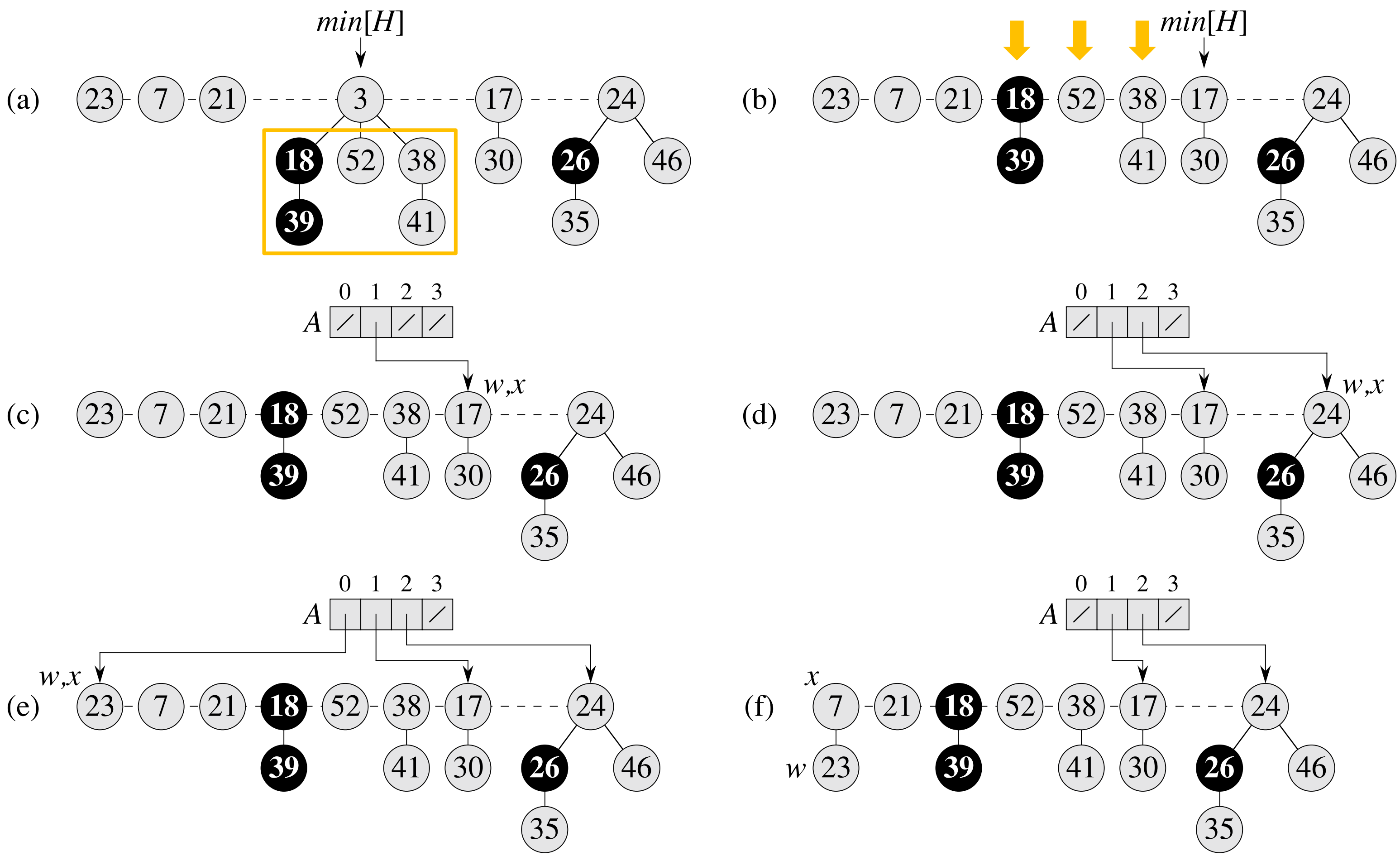
درخت‌های دارای درجه‌ی مساوی را به هم متصل می‌کند.

(دو درخت اولیه که ریشه‌ی هر کدام k فرزند دارد دارای ساختار U_k است، پس درخت حاصل ساختار U_{k+1} را دارد.)

```
FIB-HEAP-LINK(H, y, x)
remove y from the root list of H
make y a child of x, incrementing degree[x]
mark[y] ← FALSE
```

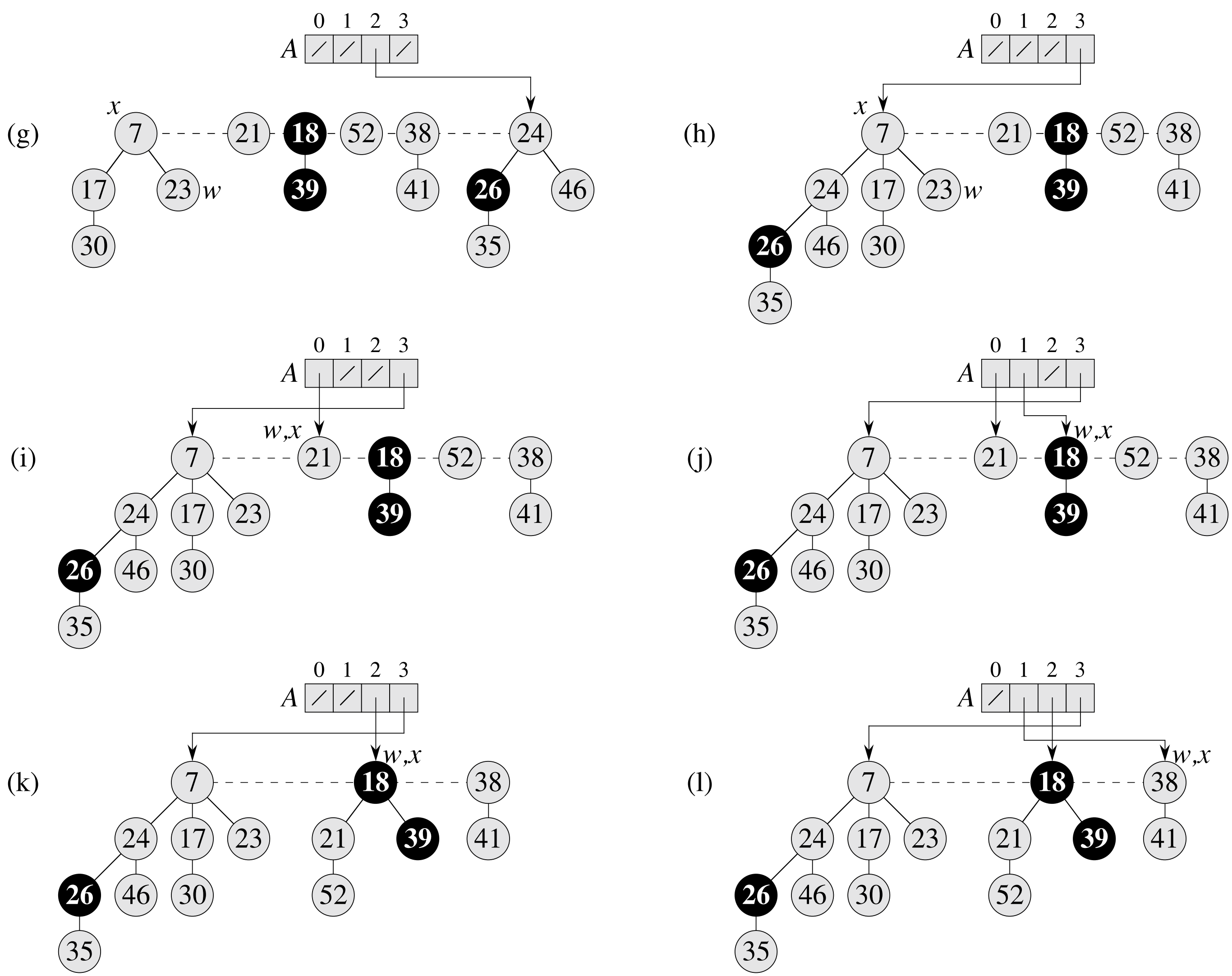
«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

استخراج گرهٔ می‌نیمم: مثال (۱ از ۳)



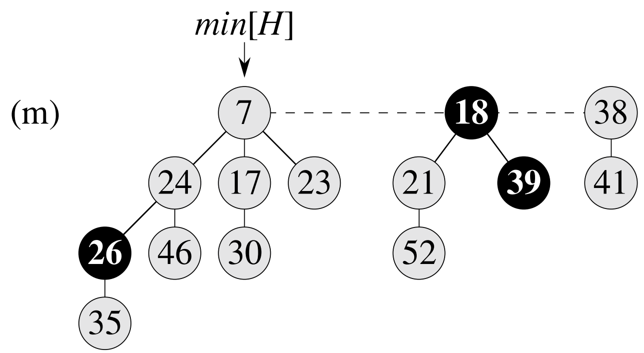
«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

استخراج گره‌ی می‌نیمم: مثال (۲ از ۳)



«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

استخراج گره‌ی می‌نیمم: مثال (۳ از ۳)



«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

استخراج گره‌ی می‌نیمم: زمان اجرای واقعی

هزینه‌ی واقعی:

$O(D(n))$ از

- وجود حداکثر $D(n)$ فرزند گره‌ی می‌نیمم پردازش شده در روال FIB-HEAP-LINK و
- اعمال خطوط (1-2,14-19) در روال CONSOLIDATE.

+ تحلیل حلقه‌ی **for** خطوط (3-13) در روال CONSOLIDATE:

اندازه‌ی لیست ریشه منهای ریشه‌ی استخراج‌شده به‌اضافه‌ی فرزندان گره‌ی استخراج‌شده:
اندازه‌ی آن حداکثر $D(n)$.

هر بار در حلقه‌ی **while** خطوط (6-12) در روال CONSOLIDATE:
یکی از ریشه‌ها به دیگری متصل می‌شود
و بنابراین مقدار کل کار انجام شده در حلقه‌ی **for** متناسب با $D(n) + t(H)$ است.

$\Leftarrow$ کل هزینه‌ی واقعی در استخراج گره‌ی می‌نیمم می‌شود: $O(D(n) + t(H))$

Prepared by Kazim Fouladi | Fall 2016 | 2nd Edition
کاظم فولادی

«عملیات هیپ ادغام‌پذیر» برای هیپ فیبوناچی

استخراج گره‌ی می‌نیمم: زمان اجرای سرشکن‌شده

هزینه‌ی سرشکن‌شده:

پتانسیل قبل از استخراج گره‌ی می‌نیمم برابر با $t(H) + 2m(H)$ است.
پتانسیل بعد از استخراج گره‌ی می‌نیمم حداکثر برابر با $(D(n) + 1) + 2m(H)$ است:
(چون حداکثر $D(n) + 1$ ریشه باقی می‌ماند و در طی عمل هیچ گره‌ای علامت نمی‌خورد.)

$\Leftarrow$ پس داریم:

$$\begin{aligned}\Delta\Phi &= \Big((D(n)+1)+2m(H)\Big) - \Big(t(H)+2m(H)\Big)\\ &= D(n)+1-t(H)\end{aligned}$$

$$c_i = O(D(n) + t(H))$$

$$\Downarrow$$

$$\hat{c}_i = c_i + \Delta\Phi = O(D(n)+t(H)) + O(D(n)) - t(H) = O(D(n))$$

$D(n) \le \lg n$

هزینه‌ی سرشکن‌شده ($\hat{c}_i$)

هزینه‌ی واقعی (c_i)

زیرا می‌توانیم واحدهای پتانسیل را بزرگ کنیم تا تأثیر مقدار ثابت موجود در $O(t(H))$ بی‌اهمیت شود.

«عملیات هیپ شامل حذف کلید» برای هیپ فیبوناچی

MERGEABLE-HEAP OPERATIONS

FIB-HEAP-DECREASE-KEY(H, x, k)

FIB-HEAP-DELETE(H, x)

«عملیات هیپ شامل حذف کلید» برای هیپ فیبوناچی

کاهش یک کلید

FIB-HEAP-DECREASE-KEY(H,x,k)

مقدار کلید x را به k کاهش می‌دهد.

```
FIB-HEAP-DECREASE-KEY(H, x, k)
if k > key[x]
    then error "new key is greater than current key"
key[x] ← k
y ← p[x]
if y ≠ NIL and key[x] < key[y]
    then CUT(H, x, y)
         CASCADING-CUT(H, y)
if key[x] < key[min[H]]
    then min[H] ← x
```

مقدار کلید جدید به x نسبت داده می‌شود.

- اگر x ریشه باشد یا مقدار x از پدرش (y) بیشتر باشد، ساختار تغییر نمی‌کند.
- وگرنه نیاز به تغییرات ساختاری داریم (**برش** و **برش آبشاری**)

«عملیات هیپ شامل حذف کلید» برای هیپ فیبوناچی

کاهش یک کلید: عملیات برش و برش آبشاری

```
CUT(H, x, y)
remove x from the child list of y, decrementing degree[y]
add x to the root list of H
p[x] ← NIL
mark[x] ← FALSE
```

اتصال x از پدرش (y) را قطع می‌کند.
x را ریشه قرار می‌دهد.
علامت x را برمی‌دارد.

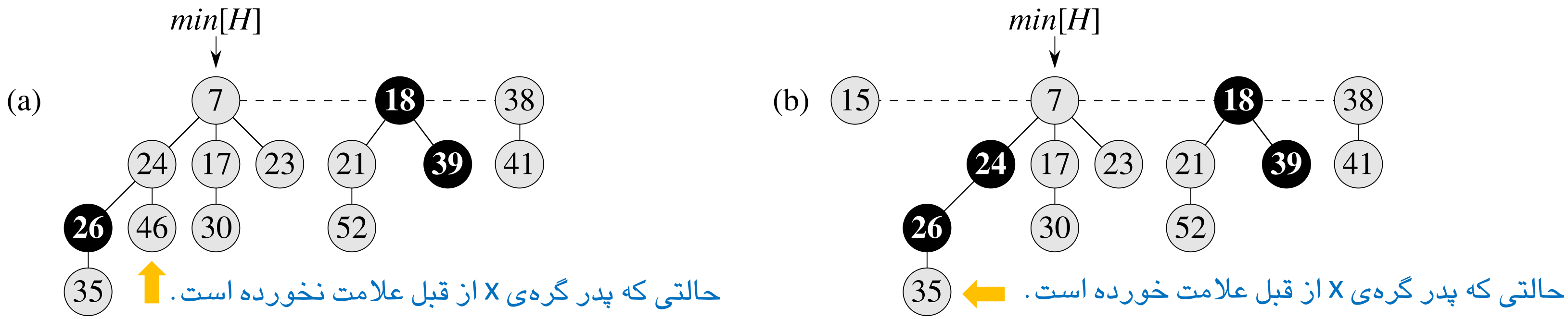
```
CASCADING-CUT(H, y)
z ← p[y]
if z ≠ NIL
   then if mark[y] = FALSE
           then mark[y] ← TRUE
           else CUT(H, y, z)
                CASCADING-CUT(H, z)
```

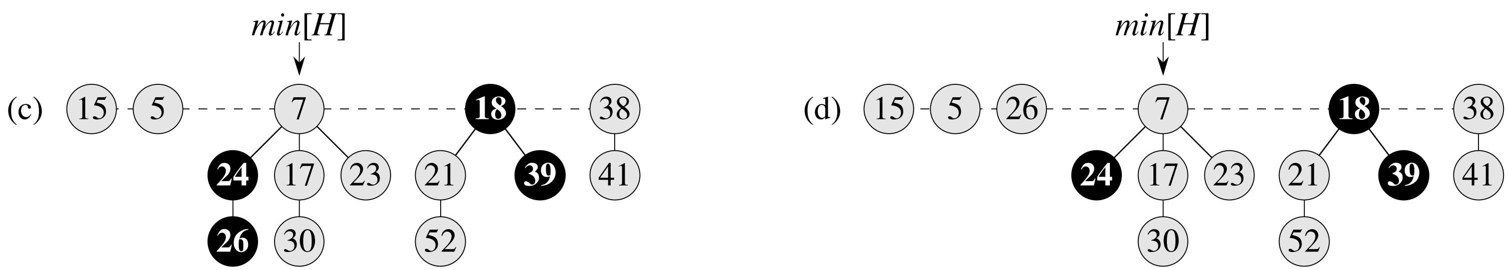
اگر y علامت نخورده باشد، علامت می‌خورد.
(چون اولین فرزندش را از دست می‌دهد.)

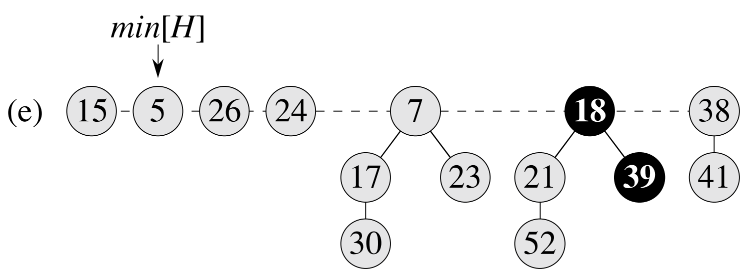
اگر y علامت خورده باشد،
(چون دومین فرزندش را از دست داده است)
به همین دلیل **برش** و **برش آبشاری** می‌خورد.
[به‌صورت بازگشتی]
این روال خودش را در راستای درخت X به سمت بالا به‌صورت بازگشتی فراخوانی می‌کند تا یک ریشه و یک گره‌ی علامت‌نخورده بیابد.

«عملیات هیپ شامل حذف کلید» برای هیپ فیبوناچی

کاهش یک کلید : مثال







Two calls of FIB-HEAP-DECREASE-KEY.

(a) The initial Fibonacci heap.

(b) The node with key 46 has its key decreased to 15. The node becomes a root, and its parent (with key 24), which had previously been unmarked, becomes marked.

(c) (c)–(e) The node with key 35 has its key decreased to 5. In part (c), the node, now with key 5, becomes a root.

رنگ سیاه = گره‌ی علامت‌دار

«عملیات هیپ شامل حذف کلید» برای هیپ فیبوناچی

کاهش یک کلید: زمان اجرای واقعی

هزینه‌ی واقعی:

$O(1)$ به‌اضافه‌ی زمان اجرای برش‌های آبشاری

به‌فرض روال $\textsc{Cascading-Cut}$ به تعداد c بار به‌طور بازگشتی فراخوانی می‌شود.
هر فراخوانی $\textsc{Cascading-Cut}$ بدون در نظر گرفتن فراخوانی‌های بازگشتی زمان $O(1)$ مصرف می‌کند.
پس هزینه‌ی واقعی کاهش کلید که همه‌ی فراخوانی‌های بازگشتی را شامل می‌شود، برابر $O(c)$ می‌شود.

«عملیات هیپ شامل حذف کلید» برای هیپ فیبوناچی

کاهش یک کلید: زمان اجرای سرشکن‌شده

هزینه‌ی سرشکن‌شده:

هر فراخوانی بازگشتی روال CASCADING-CUT به‌جز آخرین فراخوانی،
گره‌ی علامت‌خورده را جدا می‌کند و بیت علامت را پاک می‌کند:
پس، $t(H)+c$ درخت و حداکثر $m(H)-c+2$ گره‌ی علامت‌خورده وجود دارد.

$t(H)+c$ درخت = $t(H)$ درخت اولیه + $(c-1)$ درخت تولید شده در برش آبشاری + 1 درخت مشتق‌شده از x

$m(H)-c+2$ گره‌ی علامت‌خورده = $(c-1)$ گره‌ی بی‌علامت‌شده توسط برش آبشاری + احتمالاً 1 گره در آخرین فراخوانی برش آبشاری

⇐ پس داریم:

$$\begin{aligned}\Delta\Phi &= \Big(\big(t(H)+c\big)+2\big(m(H)-c+2\big)\Big)-\Big(t(H)+2m(H)\Big)\\ &= 4-c\end{aligned}$$

$$c_i = O(c)$$

$$\Downarrow$$

$$\hat{c}_i = c_i + \Delta\Phi = O(c) + (4-c) = O(1)$$

هزینه‌ی سرشکن‌شده — هزینه‌ی واقعی

دلیل انتخاب تابع پتانسیل با جمله‌ای شامل دوبرابر تعداد گره‌های علامت‌خورده:

وقتی گره‌ی علامت‌خورده‌ی y توسط یک برش آبشاری جدا می‌شود بیت علامتش پاک می‌شود، پس پتانسیل آن دو واحد کم می‌شود: یک واحد پتانسیل برای برش و پاک کردن بیت علامت و یک واحد دیگر، افزایش واحد در پتانسیل به‌واسطه‌ی ریشه شدن گره‌ی y را جبران می‌کند.

«عملیات هیپ شامل حذف کلید» برای هیپ فیبوناچی

حذف یک گره

FIB-HEAP-DELETE(H,x)

کلید x را از هیپ فیبوناچی حذف می‌کند.

```
FIB-HEAP-DELETE(H, x)
1   FIB-HEAP-DECREASE-KEY(H, x, −∞)
2   FIB-HEAP-EXTRACT-MIN(H)
```

$$\hat{c}_i = \underbrace{O(1)}_{\text{کاهش کلید}} + \underbrace{O(D(n))}_{\text{استخراج می‌نیمم}}$$

$D(n) \leq \lg n$

$$\hat{c}_i = O(\lg n))$$

هیپ فیبوناچی

قرار دادن کران بر روی ماکزیمم درجه

For each node x within a Fibonacci heap,
define $\text{size}(x)$ to be the number of nodes, including x itself, in the subtree rooted at x

Lemma 20.3
Let x be any node in a Fibonacci heap, and let $k = \mathit{degree}[x]$. Then, $\text{size}(x) \geq F_{k+2} \geq \phi^k$, where $\phi = (1+\sqrt{5})/2$.

Corollary 20.4
The maximum degree $D(n)$ of any node in an n-node Fibonacci heap is $O(\lg n)$.

عملیات بر روی هیپ فیبوناچی

خلاصه‌ی زمان‌های اجرا

Procedure	Binary heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$O(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$O(\lg n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$O(\lg n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$