Lecture 9.1: Fibonacci heap

- **Định nghĩa**

  Một *Fibonacci heap* là một tập các cây mà mỗi cây đều là heap-ordered.

  – Cây trong Fibonacci heap không cần thiết phải là cây nhị thức.

  – Cây trong Fibonacci heap là có gốc nhưng không có thứ tự (unordered).

- Ứng dụng của Fibonacci heap trong các bài toán trên đồ thị:

  – Giải thuật Prim để xác định một cây khung nhỏ nhất trong một đồ thị có trọng số.(Tìm theo lân cận gần)

  – Giải thuật Dijkstra để tìm một đường đi ngắn nhất trong đồ thị có hướng và có trọng số dương.(Dựa trên phương pháp gán nhãn tạm thời cho các đỉnh)

Cấu trúc của Fibonacci heap

Mỗi nút *x* có các trường

- *p*[*x*]: con trỏ đến nút cha của nó.
- *child*[*x*]: con trỏ đến một con nào đó trong các con của nó.
  - Các con của *x* được liên kết với nhau trong một danh sách vòng liên kết kép (circular, doubly linked list), gọi là *danh sách các con* của *x*.
  - Mỗi con *y* trong danh sách các con của *x* có các con trỏ
    - *left*[*y*], *right*[*y*] chỉ đến các anh em bên trái và bên phải của *y*.

      Nếu *y* là con duy nhất của *x* thì *left*[*y*] = *right*[*y*] = *y*.

Cấu trúc của Fibonacci heap

(tiếp)

Các trường khác trong nút *x*

– *degree*[*x*]: số các nút con chứa trong danh sách các nút con của nút *x*

– *mark*[*x*]: có trị bool là TRUE hay FALSE,

  - chỉ rằng *x* có bị xóa một nút con (*mark*[*x*] = TRUE) hay không kể từ lần cuối mà *x* được làm thành nút con của một nút khác.
  - *mark*[*x*] = TRUE – tương đương với 1 đơn vị trả trước cho các thao tác tiếp theo (như: đưa nút này lên danh sách gốc, hiệu chỉnh consolidate)
  - Nút *x* được tô đen ⇔ *mark*[*x*] = TRUE.

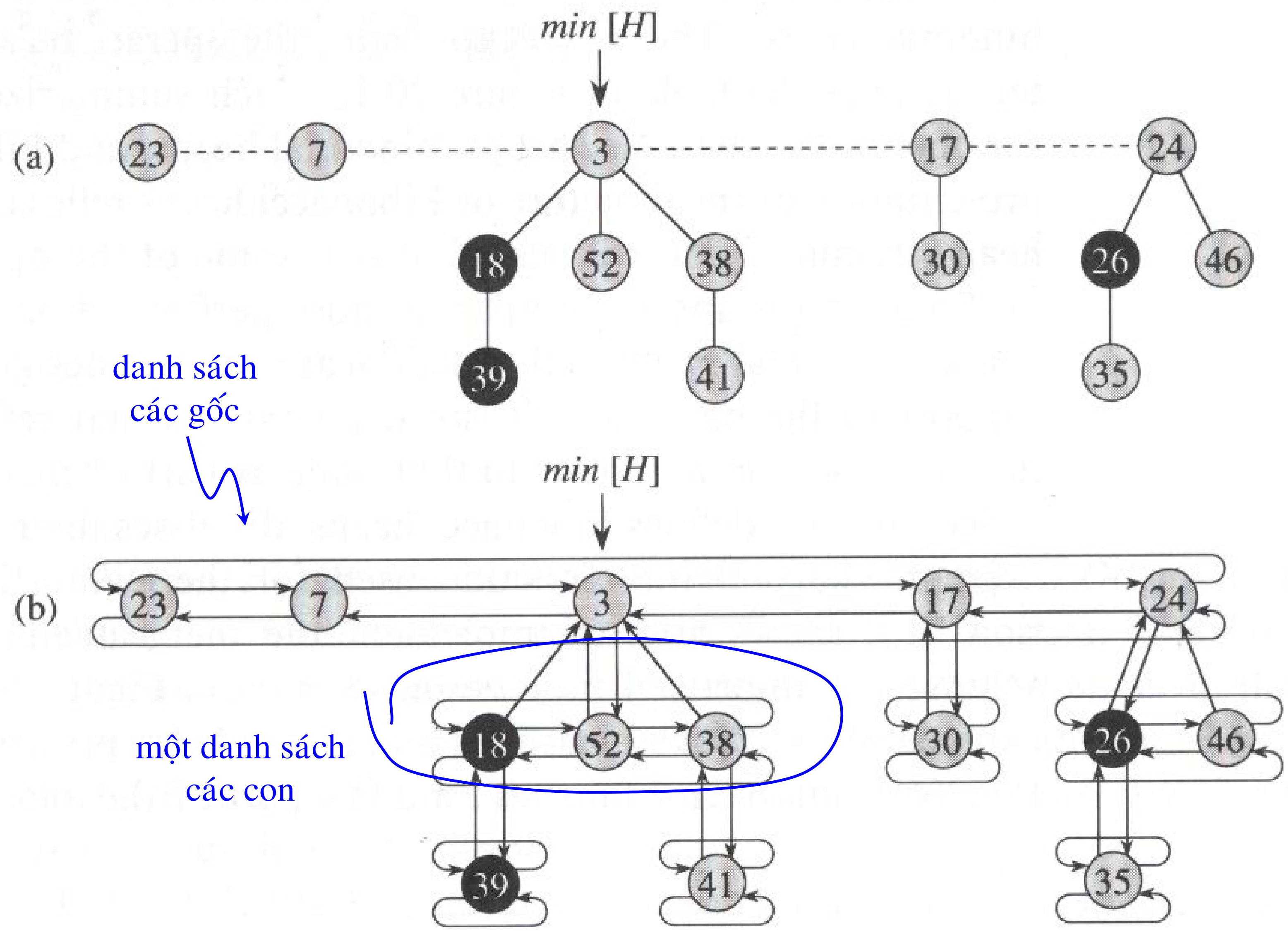
Cấu trúc của Fibonacci heap

(tiếp)

Nếu *H* là Fibonacci heap

- Truy cập *H* bằng con trỏ *min*[*H*] đến nút gốc của cây chứa khoá nhỏ nhất gọi là *nút nhỏ nhất* của *H*.
  - Nếu *H* là trống thì *min*[*H*] = NIL.
- Tất cả các nút gốc của các cây trong *H* được liên kết với nhau bởi các con trỏ *left* và *right* của chúng thành một danh sách liên kết kép vòng gọi là *danh sách các gốc* của *H*.
- *n*[*H*]: số các nút hiện có trong *H*.

Cấu trúc của Fibonacci heap: ví dụ



Hàm thế năng

- Dùng phương pháp thế năng để phân tích hiệu suất của các thao tác lên một Fibonacci heap.
- Cho một Fibonacci heap H
  - gọi số các cây của Fibonacci heap H là $t(H)$
  - gọi số các nút x được đánh dấu ($mark[x]$ = TRUE) là $m(H)$.

  Hàm thế năng của H được định nghĩa như sau
  - $\boxed{\Phi(H) = t(H) + 2\, m(H)}$
  - thế năng của một tập các Fibonacci heap là tổng của các thế năng của các Fibonacci heap thành phần.

Hàm thế năng (tiếp)

Khi bắt đầu hàm thế năng có trị là 0, sau đó hàm thế năng có trị ≥ 0. Do đó chi phí khấu hao tổng cộng là một cận trên của chi phí thực sự tổng cộng cho dãy các thao tác.

- Lưu ý: một đơn vị thế năng sẽ được dùng để "trả" cho một lượng cố định công việc. Ta sẽ xác định lượng công việc này sau.

Bậc tối đa

- Gọi $D(n)$ là cận trên cho bậc lớn nhất của một nút bất kỳ trong một Fibonacci heap có $n(H)$ nút.
- $D(n) = O(\lg n)$.

Các thao tác lên heap hợp nhất được

- Nếu chỉ dùng các thao tác lên heap hợp nhất được:
  - MAKE-HEAP
  - INSERT
  - MINIMUM
  - EXTRACT-MIN
  - UNION

  thì mỗi Fibonacci heap là một tập các cây nhị thức "*không thứ tự* ".

Cây nhị thức không thứ tự

- Một *cây nhị thức không thứ tự* (unordered binomial tree) được định nghĩa đệ quy như sau
  – Cây nhị thức không thứ tự U_0 gồm một nút duy nhất.
  – Một cây nhị thức không thứ tự U_k được tạo bởi hai cây nhị thức không thứ tự U_{k-1} bằng cách lấy gốc của cây này làm nút con (vị trí trong danh sách các con là tùy ý) của gốc của cây kia.
- Lemma các tính chất của một cây nhị thức cũng đúng cho các cây nhị thức không thứ tự, nhưng với thay đổi sau cho tính chất 4:

  4'. Đối với cây nhị thức không thứ tự U_k , nút gốc có bậc là k, giá trị k lớn hơn bậc của mọi nút bất kỳ khác. Các con của gốc là gốc của các cây con U_0 , U_1 ,..., U_{k-1} trong một thứ tự nào đó.

Tạo một Fibonacci heap mới

- Thủ tục để tạo một Fibonacci heap trống:
  MAKE-FIB-HEAP
  - cấp phát và trả về đối tượng Fibonacci heap *H*, với
    $n[H] = 0$,
    $min[H] = \text{NIL}$
- Phân tích thủ tục MAKE-FIB-HEAP
  - Chi phí thực sự là $O(1)$
  - Thế năng của Fibonacci heap rỗng là
    $\Phi(H) = t(H) + 2\, m(H)$
    $= 0$.
  - Vậy chi phí khấu hao là $O(1)$. ($\hat{C}_i = C_i + \Phi_i(H) - \Phi_{i-1}(H)$)

Chèn một nút vào Fibonacci heap

- Thủ tục để chèn một nút vào một Fibonacci heap:

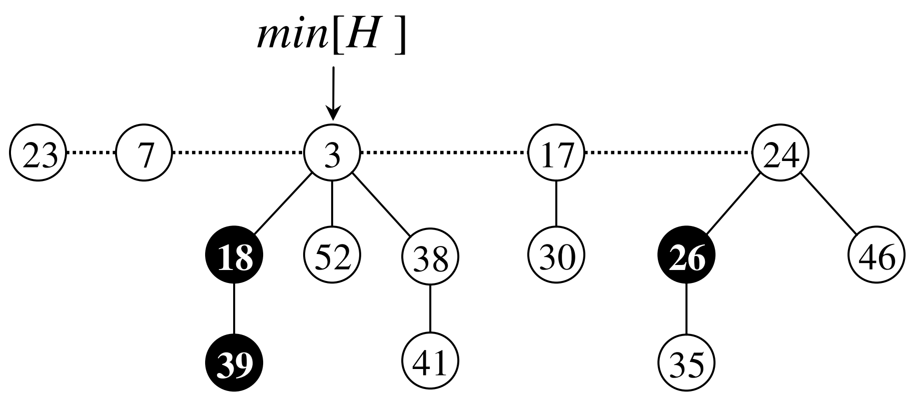
  FIB-HEAP-INSERT

  – chèn nút x (mà $key[x]$ đã được gán trị) vào Fibonacci heap H

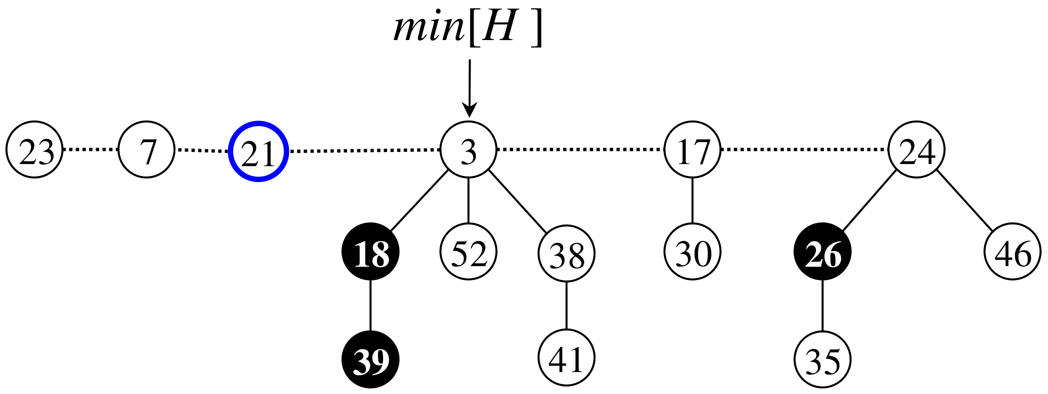
```
FIB-HEAP-INSERT(H, x)
degree[x] ← 0
p[x] ← NIL
child[x] ← NIL
left[x] ← x
right[x] ← x
mark[x] ← FALSE
nối x vào danh sách các gốc của H
if min[H] = NIL or key[x] < key[min[H]]
   then min[H] ← x
n[H] ← n[H] + 1
```

Ví dụ chèn một nút vào Fibonacci heap

(tiếp)



FIB-HEAP-INSERT(*H*, *x*), với *key*[*x*] = 21



Chèn một nút vào Fibonacci heap (tiếp)

- Phân tích thủ tục FIB-HEAP-INSERT:

  Phí tổn khấu hao là $O(1)$ vì

  – Gọi H là Fibonacci heap đầu vào, và H' là Fibonacci heap kết quả.

  – Ta có: $t(H') = t(H) + 1$,

  $m(H') = m(H)$.

  Vậy hiệu thế $\Phi(H') - \Phi(H)$ bằng

  $((t(H) + 1) + 2m(H)) - (t(H) + 2m(H)) = 1$.

  – Phí tổn khấu hao bằng

  phí tổn thực sự + hiệu thế $= O(1) + 1$

  $= O(1)$.

Tìm nút nhỏ nhất trong Fibonacci Heap

- Con trỏ *min*[*H*] chỉ đến nút nhỏ nhất của Fibonacci heap *H*.
- Phân tích:
  - Phí tổn thực sự là $O(1)$
  - Hiệu thế là 0 vì thế năng của *H* không thay đổi
  - Vậy phí tổn khấu hao là $O(1)$ (= phí tổn thực sự).

Hợp nhất hai Fibonacci heap

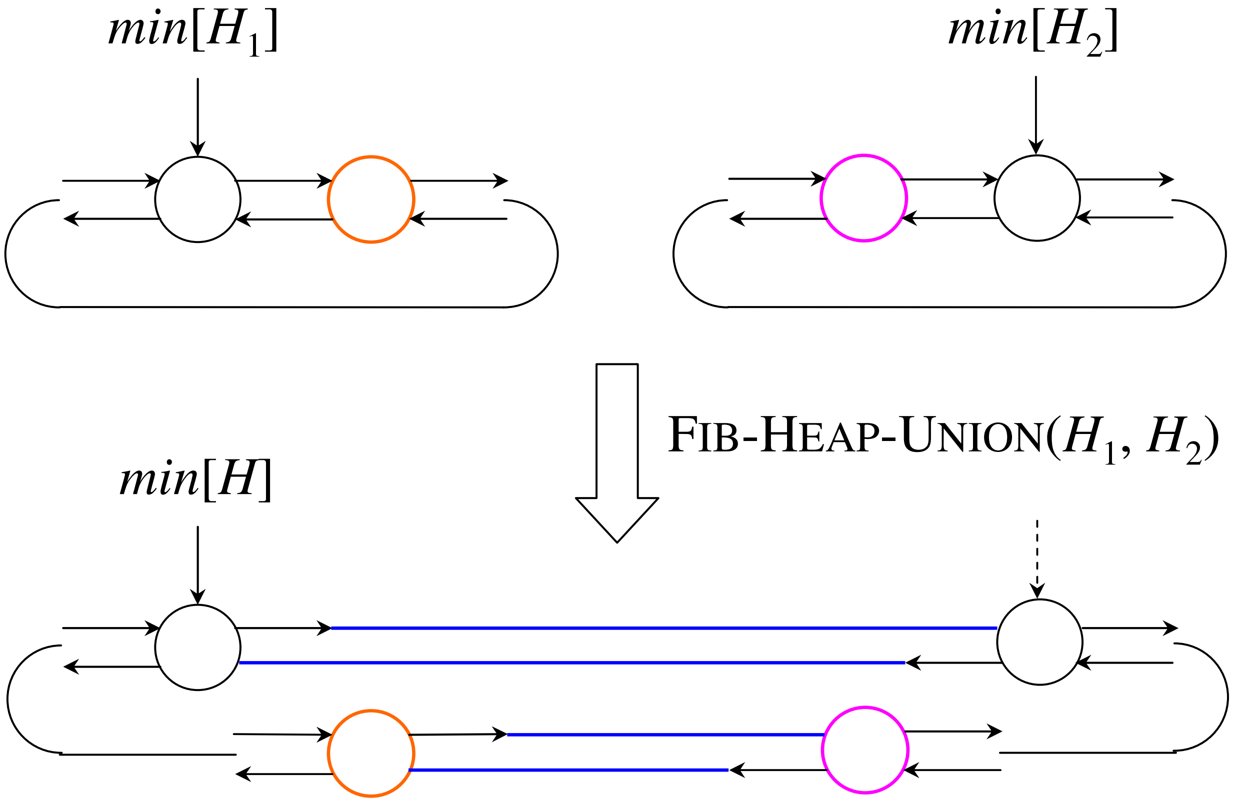
- Thủ tục để hợp nhất hai Fibonacci heap:
  FIB-HEAP-UNION
  - hợp nhất các Fibonacci heap H_1 và H_2
  - trả về H - Fibonacci heap kết quả

FIB-HEAP-UNION(H_1, H_2)
$H \leftarrow$ MAKE-FIB-HEAP()
$min[H] \leftarrow min[H_1]$
nối danh sách các gốc của H_2 với danh sách các gốc của H
if ($min[H_1]$ = NIL) or
  ($min[H_2] \neq$ NIL and $key[min[H_2]] < key[min[H_1]]$)
  then $min[H] \leftarrow min[H_2]$
$n[H] \leftarrow n[H_1] + n[H_2]$
giải phóng (free) các đối tượng H_1 và H_2
return H

Hợp nhất hai Fibonacci heap

(tiếp)

- Ví dụ: giả sử $key[min[H_1]] < key[min[H_2]]$



Hợp nhất hai Fibonacci heap (tiếp)

- Phân tích thủ tục FIB-HEAP-UNION:

  Phí tổn khấu hao được tính từ

  – phí tổn thực sự là $O(1)$

  – hiệu thế là

  $$\Phi(H) - (\Phi(H_1) + \Phi(H_2))$$
  $$= (t(H) + 2m(H)) - ((t(H_1) + 2m(H_1)) + (t(H_2) + 2m(H_2)))$$
  $$= 0, \quad \text{vì} \quad t(H) = t(H_1) + t(H_2) \text{ và}$$
  $$m(H) = m(H_1) + m(H_2)$$

  – Vậy phí tổn khấu hao = phí tổn thực sự + hiệu thế

  $$= O(1)$$

Tách lấy ra nút nhỏ nhất

- Thủ tục để tách ra nút nhỏ nhất:

  FIB-HEAP-EXTRACT-MIN

  – tách nút nhỏ nhất khỏi Fibonacci heap *H*

```
FIB-HEAP-EXTRACT-MIN(H)
    z ← min[H]
    if z ≠ NIL
       then for với mỗi nút con x của  z
                    do thêm x vào danh sách các gốc của H
                       p[x] ← NIL
            lấy z ra khỏi danh sách các gốc của H
            if z = right[z]
               then min[H] ← NIL
               else min[H]  ← right[z]
                    CONSOLIDATE(H)
            n[H] ← n[H] – 1
    return z
```

Củng cố (consolidate)

```
CONSOLIDATE(H)
for i ← 0 to D(n[H])
     do A[i] ← NIL
for với mỗi nút w trong danh sách các gốc của H
     do x ← w
        d ← degree[x]
        while A[d] ≠ NIL
            do y ← A[d]
               if key[x] > key[y]
                  then đổi chỗ 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 //thêm A[i] vào danh sách các gốc của H
                if min[H] = NIL or key[A[i]] < key[min[H]]
                   then min[H] ← A[i]
```

Củng cố (consolidate)

Thủ tục phụ: *củng cố* danh sách các gốc của một Fibonacci heap H

- liên kết mọi cặp gốc có cùng bậc thành một gốc mới cho đến khi mọi gốc có bậc khác nhau.
- $A[0 \,.\,.\, D(n[H])]$

Liên kết hai gốc có cùng bậc

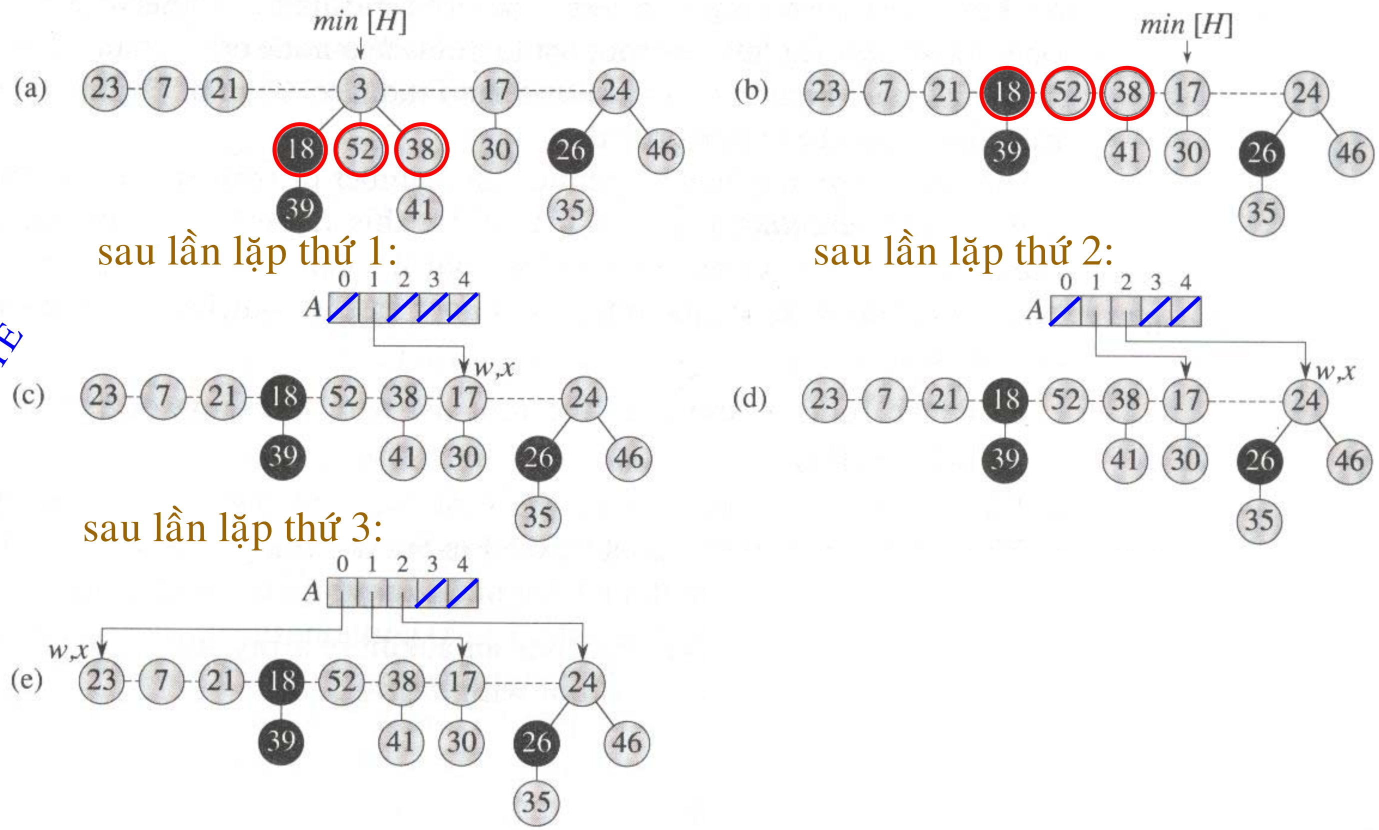
– Thủ tục CONSOLIDATE liên kết các gốc có cùng bậc cho đến khi mọi gốc có được sau đó đều có bậc khác nhau.

- Dùng thủ tục FIB-HEAP-LINK(H, y, x) để tách gốc y khỏi danh sách gốc của H, sau đó liên kết gốc y vào gốc x, gốc x và gốc y có cùng bậc.

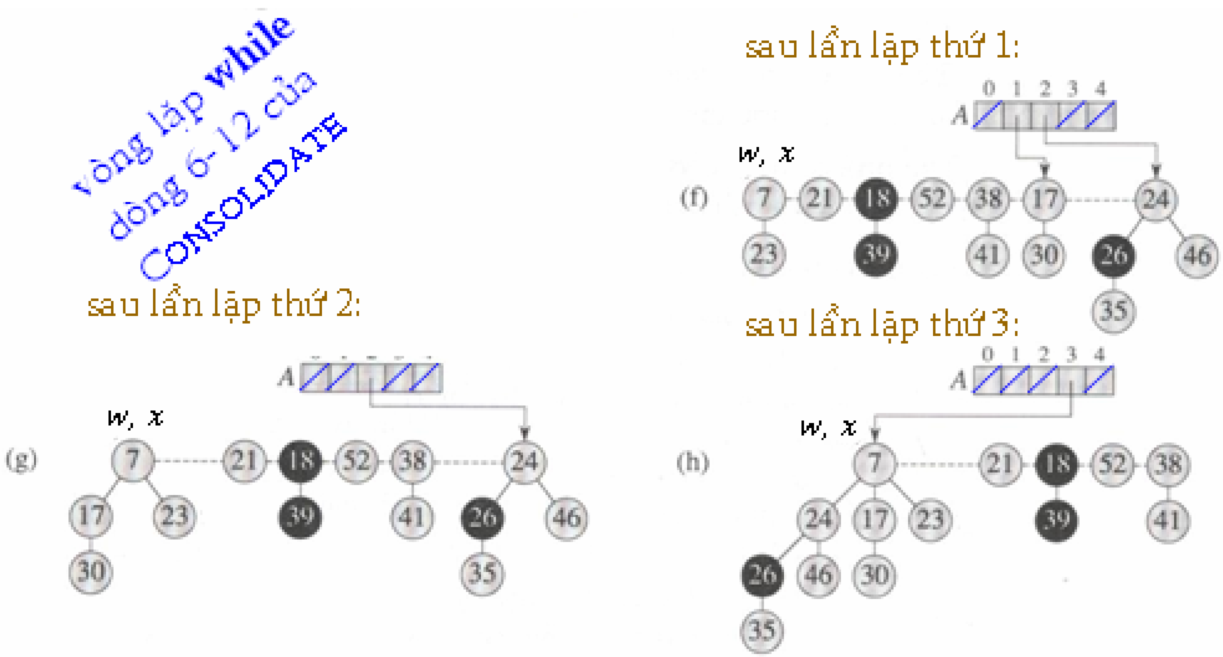
```
FIB-HEAP-LINK(H, y, x)
    lấy y ra khỏi danh sách các gốc của H
    y thành nút con của x, tăng degree[x]
    mark[y] ← FALSE
```

Thực thi Fib-Heap-Extract-Min: ví du (A[0-3])

vòng lặp **for** dòng 3-13 của CONSOLIDATE



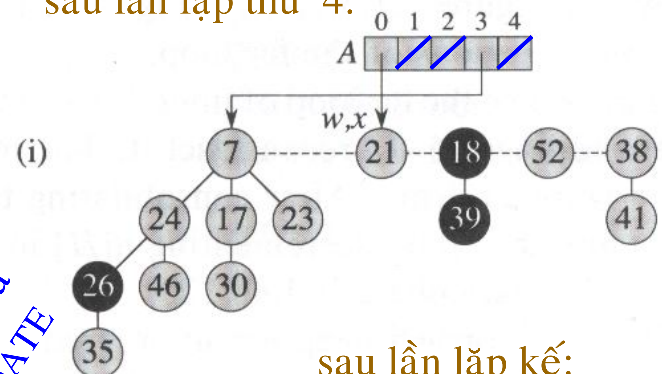
Thực thi FIB-HEAP-EXTRACT-MIN: ví dụ



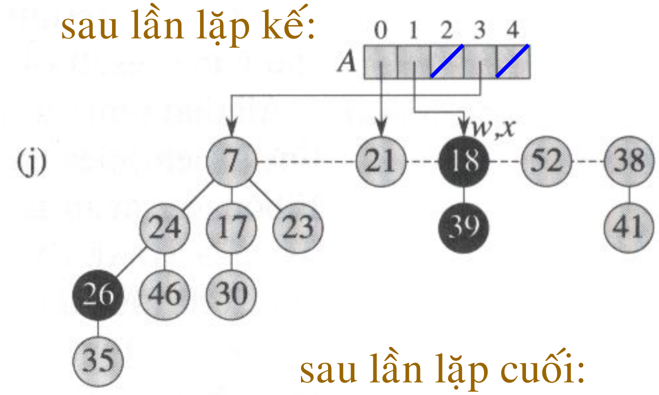
Thực thi FIB-HEAP-EXTRACT-MIN: ví dụ (tiếp)

vòng lặp **for** dòng 3-13 của CONSOLIDATE

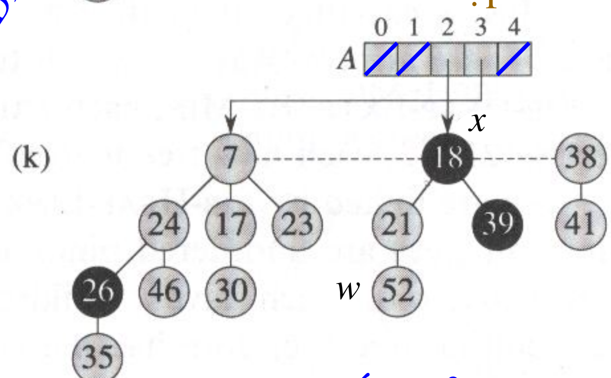
sau lần lặp thứ 4:



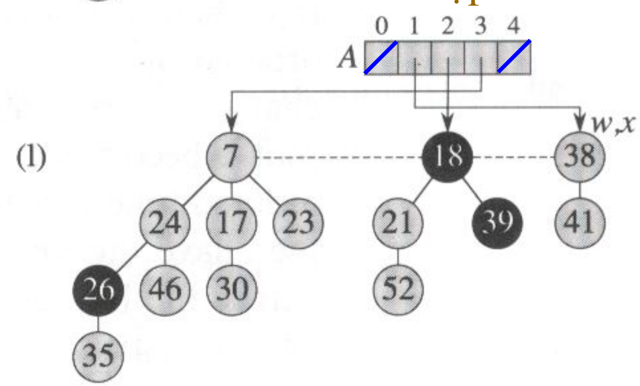
sau lần lặp kế:



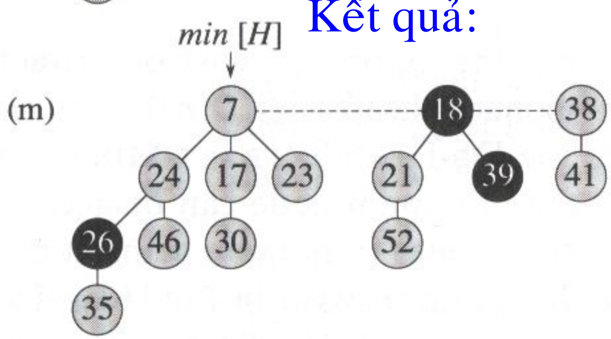
sau lần lặp kế:



sau lần lặp cuối:



Kết quả:



Chi phí thực sự của FIB-HEAP-EXTRACT-MIN

- Gọi *H* là Fibonacci heap ngay trước khi gọi FIB-HEAP-EXTRACT-MIN, số nút của *H* là $n(H)$.
  - Chi phí thực sự bao gồm:
    - $O(D(n))$: vì có nhiều nhất là $D(n)$ nút con của nút nhỏ nhất cần được xử lý bởi:
      - FIB-HEAP-EXTRACT-MIN
      - các dòng 1-2 và 14-19 của CONSOLIDATE
    - $O(D(n) + t(H))$: vì khi gọi CONSOLIDATE chiều dài của danh sách gốc nhiều nhất là $D(n) + t(H) - 1$, mà thời gian chạy vòng lặp **for** dòng 3-13 nhiều nhất là tỉ lệ với chiều dài của danh sách gốc này.
  - Vậy chi phí thực sự của FIB-HEAP-EXTRACT-MIN là $O(D(n) + t(H))$.

Chi phí khấu hao của FIB-HEAP-EXTRACT-MIN

- Gọi H' là Fibonacci heap sau khi gọi FIB-HEAP-EXTRACT-MIN lên H.
  - Nhắc lại: hàm thế năng của H được định nghĩa là

    $$\Phi(H) = t(H) + 2\, m(H)$$
  - Biết:

    chi phí khấu hao = chi phí thực sự + $\Phi(H') - \Phi(H)$
    - Đã tính: phí tổn thực sự của FIB-HEAP-EXTRACT-MIN là $O(D(n) + t(H))$.
- Sau khi gọi FIB-HEAP-EXTRACT-MIN lên H, số gốc (hay số cây) của H' nhiều nhất là $D(n) + 1$, và không có nút nào được đánh dấu thêm. Vậy:

$$\Phi(H') = (D(n) + 1) + 2\, m(H') \qquad \text{với } (m(H') <= \mathrm{m(H)})$$

Chi phí khấu hao của FIB-HEAP-EXTRACT-MIN

(tiếp)

– Do đó chi phí khấu hao của FIB-HEAP-EXTRACT-MIN là

$$O(D(n) + t(H)) + ((D(n) + 1) + 2\, m'(H)) - (t(H) + 2\, m(H))$$

$$= O(D(n)) + O(t(H)) - \underbrace{t(H)}_{\text{đến từ thế năng}}$$

$$= O(D(n)) \,,$$

Giảm khóa của một nút

- Thủ tục để giảm khóa của một nút:

  FIB-HEAP-DECREASE-KEY

  – giảm khóa của nút *x* trong Fibonacci heap *H* thành trị mới *k* nhỏ hơn trị cũ của khóa.

```
FIB-HEAP-DECREASE-KEY(H, x, k)
if k > key[x]
   then error "khóa mới lớn hơn khóa hiện thời"
key[x] ← k
y ← p[x]
if y ≠ NIL and key[x] < key[y]
   then CUT(H, x, y) – cắt nút x lên danh sách gốc
        CASCADING-CUT(H, y)
if key[x] < key[min[H]]
   then min[H] ← x
```

Giảm khóa của một nút (tiếp)

- Thủ tục phụ để cắt liên kết giữa nút con *x* và nút cha y, sau đó làm *x* thành một gốc mới.

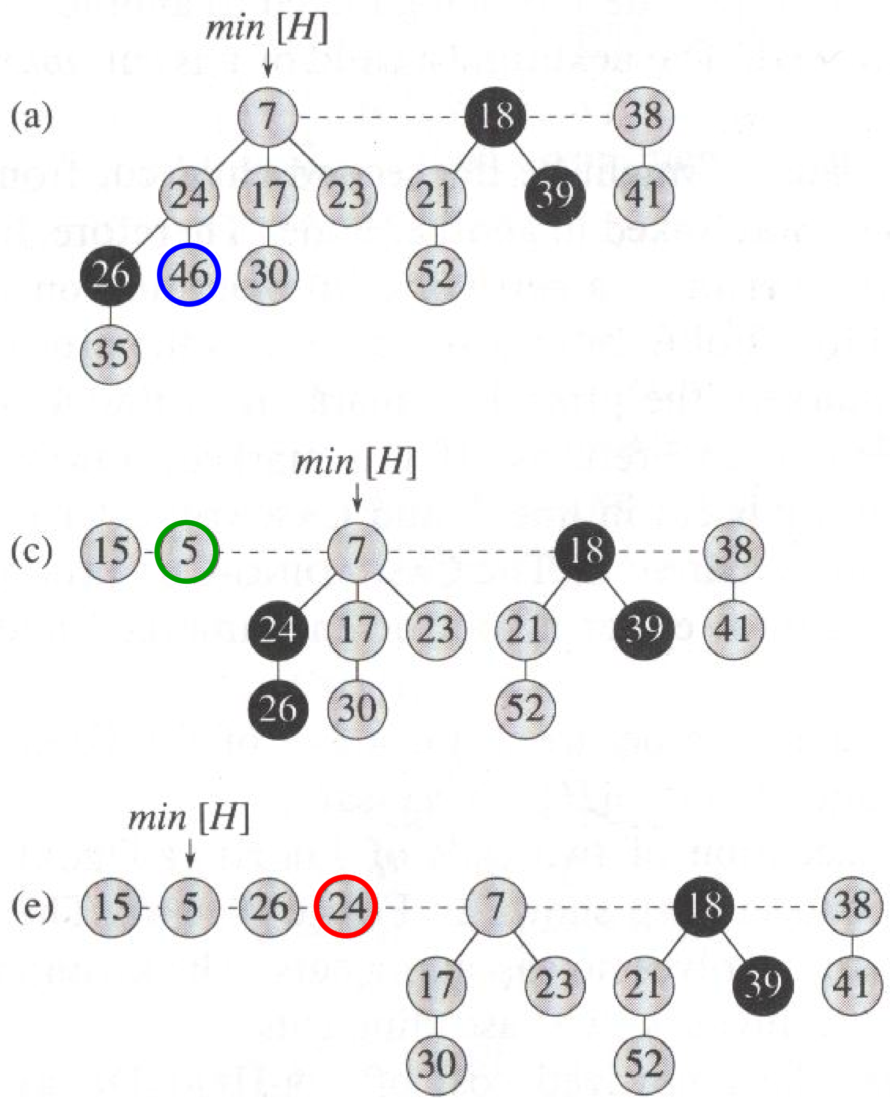
```
CUT(H, x, y)
    lấy x ra khỏi danh sách các nút con của y, giảm degree[y]
    thêm x vào danh sách các gốc của H
    p[x] ← NIL
    mark[x] ← FALSE
```

Giảm khóa của một nút (tiếp)

- Thủ tục phụ để xử lý nút cha **y** của nút bị cắt **x** dựa trên trường *mark*[*y*].

```
CASCADING-CUT(H, y)
  z ← p[y]
  if z ≠ NIL –chưa đi đến gốc
     then if mark[y] = FALSE
             then mark[y] ← TRUE
             else CUT(H, y, z) – cắt nút y đưa lên danh sách gốc
                  CASCADING-CUT(H, z)
```

Giảm khoá của một nút: ví dụ

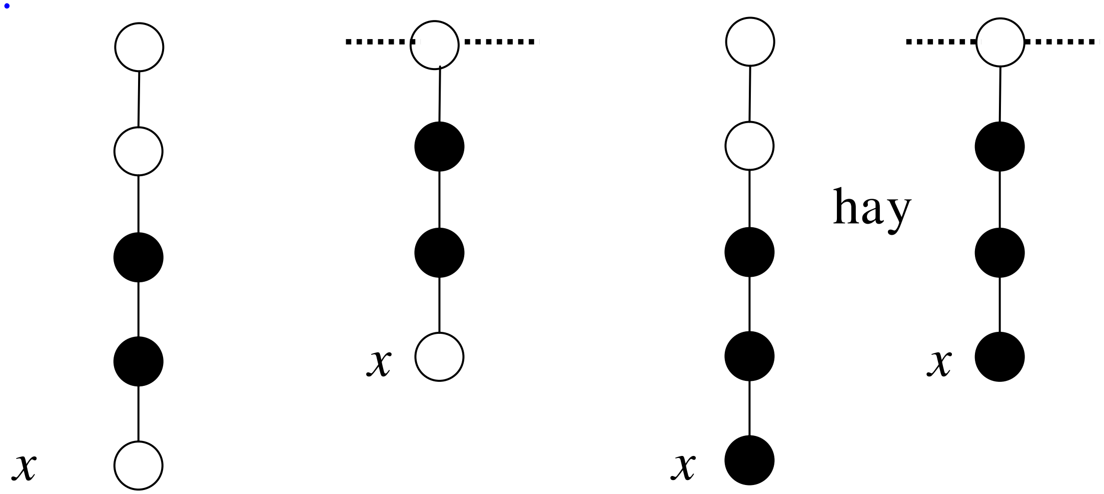


(a) Heap ban đầu
(b) Giảm khóa 46 thành 15
(c)-(e) Giảm khóa 35 thành 5

Chi phí thực sự của FIB-HEAP-DECREASE-KEY

- Gọi *H* là Fibonacci heap ngay trước khi gọi FIB-HEAP-DECREASE-KEY, số nút của *H* là *n (H)*.
  - Chi phí thực sự của FIB-HEAP-DECREASE-KEY bao gồm:
    - $O(1)$: dòng 1-5 và 8-9,
    - thời gian thực thi các cascading cut. Giả sử CASCADING-CUT được gọi đệ quy *c* lần. Thời gian thực thi CASCADING-CUT là $O(1)$ không kể các gọi đệ quy.

Ví dụ:



Vậy phí tổn thực sự của FIB-HEAP-DECREASE-KEY là $O(c)$.

Chi phí khấu hao của FIB-HEAP-DECREASE-KEY

- Gọi *H'* là Fibonacci heap sau khi gọi FIB-HEAP-DECREASE-KEY lên *H*.
  - Nhắc lại: hàm thế năng của *H* được định nghĩa là

    $\Phi(H) = t(H) + 2\, m(H)$
  - chi phí khấu hao = chi phí thực sự + $\Phi(H') - \Phi(H)$
    - Đã tính: chi phí thực sự của FIB-HEAP-DECREASE-KEY là $O(c)$.
    - Sau khi gọi FIB-HEAP-DECREASE-KEY lên *H*, thì *H'* có $t(H) + c$ cây.

    $\Phi(H') - \Phi(H) \le (t(H) + c) + 2(m(H) - (c - 1) + 1)) - (t(H) + 2\, m(H))$

    $\le 4 - c.$

số lần gọi CUT bằng số lần gọi CASCADING-CUT = c, mà
– mỗi lần thực thi CUT thì 1 nút trở thành cây
– mỗi lần thực thi CASCADING-CUT ngoại trừ lần cuối của gọi đệ quy thì 1 nút được unmarked và lần cuối của gọi đệ quy CASCADING-CUT có thể marks 1 nút.

Chi phí khấu hao của FIB-HEAP-DECREASE-KEY

(tiếp)

– Do đó chi phí khấu hao của FIB-HEAP- DECREASE-KEY là

$$O(c) + \underbrace{4 - c}_{\text{đến từ thế năng}} = O(1),$$

vì có thể tăng đơn vị của thế năng để khống chế hằng số ẩn trong $O(c)$.

Xóa một nút

- Thủ tục để xóa một nút:

  FIB-HEAP-DELETE

  – Xóa một nút x khỏi một Fibonacci heap H.

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

MST-Prim (MINIMUM SPANNING TREE)

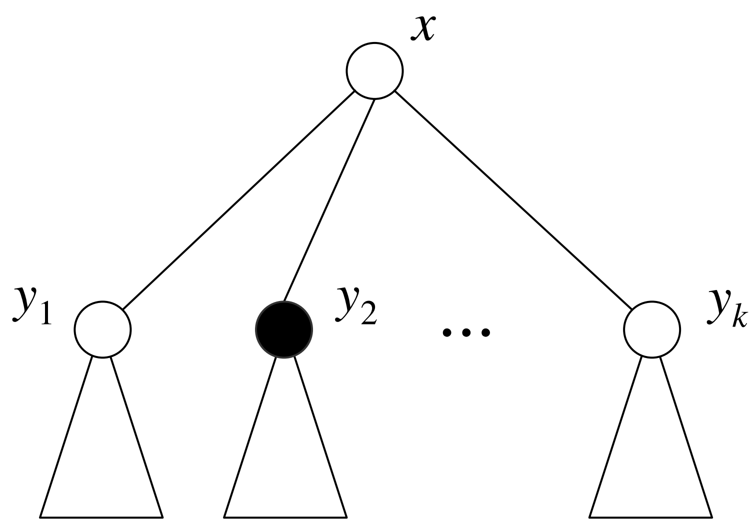
MST_PRIM (G, *w*, *v*)

```
1.  Q ← V[G]
2.  for each u in Q do
3.      key [u] ← ∞
4.  key [r] ← 0
5.  π[r] ← NIL
6.  while queue is not empty do
7.      u ← EXTRACT_MIN (Q)
8.      for each v in Adj[u] do
9.          if v is in Q and w(u, v) < key [v]
10.             then π[v] ← u
11.                 key [v] ← w(u, v)
```

Q is implemented as a Fibonacci heap

- **Thực hiện DECREASE KEY trong dòng 11.**

Chặn trên lên bậc lớn nhất



Lemma (Bổ đề 21.1)

Cho x là một nút bất kỳ trong một Fibonacci heap, và giả sử $degree[x] = k$. Gọi $y_1, y_2,..., y_k$ là các con của x được xếp theo thứ tự lúc chúng được liên kết vào x, từ lúc sớm nhất đến lúc trễ nhất. Thì $degree[y_1] \geq 0$ và $degree[y_i] \geq i - 2$ với $i = 2, 3,..., k$.

Chặn trên lên bậc lớn nhất

(tiếp)

Chứng minh

– Rõ ràng là $degree[y_1] \geq 0$.

$i \geq 2$:

– Khi y_i được liên kết vào x thì $y_1, y_2, \ldots, y_{i-1}$ là trong tập các con của x nên khi đó $degree[x] \geq i - 1$.

- Nút y_i được liên kết vào x chỉ khi nào $degree[x] = degree[y_i]$, vậy khi đó $degree[y_i]$ cũng $\geq i - 1$.

– Kể từ khi đó thì nút y_i mất nhiều nhất là một nút con, vì nếu nó mất hai nút con thì nó đã bị cắt khỏi x. Vậy

$$\begin{aligned} degree[y_i] &\geq (i - 1) - 1 \\ &\geq i - 2 . \end{aligned}$$

Chặn trên lên bậc lớn nhất (tiếp)

Định nghĩa

Với $k = 0, 1, 2, \ldots$ định nghĩa F_k là *số Fibonacci thứ k*:

$$F_k = \begin{cases} 0 & \text{nếu } k = 0, \\ 1 & \text{nếu } k = 1, \\ F_{k-1} + F_{k-2} & \text{nếu } k \geq 2. \end{cases}$$

Lemma (Lemma 21.2, bài tập)

Với mọi số nguyên $k \geq 0$,

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i \ .$$

Theo 2.2-8:

Với mọi số nguyên $k \geq 0$, ta có $F_{k+2} \geq \phi^k$, trong đó $\phi = (1 + \sqrt{5}) / 2$, *tỉ số vàng*.

Lemma 21.2

For all integers $k \geq 0$,

$$F_{k+2} = 1 + \sum_{i=0}^{k} F_i \ .$$

Proof The proof is by induction on k. When $k = 0$,

$$\begin{aligned} 1 + \sum_{i=0}^{0} F_i &= 1 + F_0 \\ &= 1 + 0 \\ &= 1 \\ &= F_2 \ . \end{aligned}$$

We now assume the inductive hypothesis that $F_{k+1} = 1 + \sum_{i=0}^{k-1} F_i$, and we have

$$\begin{aligned} F_{k+2} &= F_k + F_{k+1} \\ &= F_k + \left(1 + \sum_{i=0}^{k-1} F_i\right) \\ &= 1 + \sum_{i=0}^{k} F_i \ . \end{aligned}$$

Chặn trên lên bậc lớn nhất (tiếp)

Với mọi nút x trong một Fibonacci heap, định nghĩa size(x) là số nút, kể cả x, trong cây con có gốc là x.

Lemma (Lemma 21.3)

Cho x là một nút bất kỳ trong một Fibonacci heap, và cho k = *degree*[x]. Thì size(x) $\geq F_{k+2} \geq \phi^k$, trong đó $\phi = (1 + \sqrt{5}) / 2$.

Chứng minh

- Gọi s_k là trị nhỏ nhất có thể được của size(z) trên mọi nút z với *degree*[z] = k trong heap Fibonacci bất kỳ.
- Rõ ràng là $s_0 = 1$, $s_1 = 2$.
- Ta có $s_k \leq$ size(x)

Chặn trên lên bậc lớn nhất

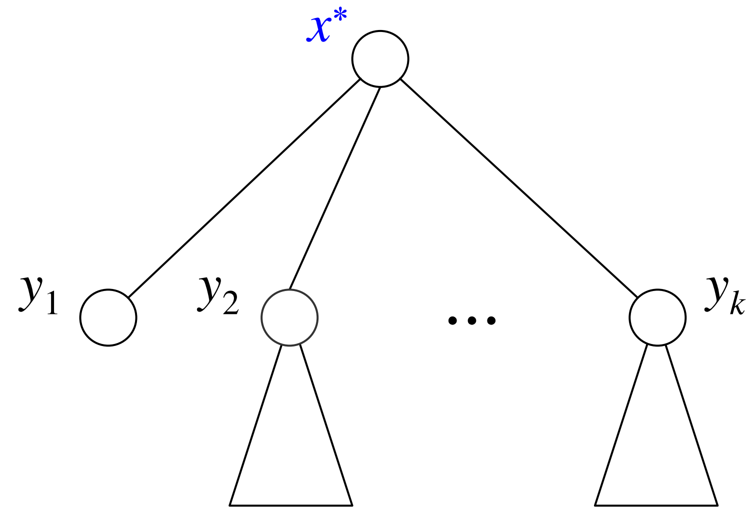
Chứng minh (tiếp)

$$\text{size}(x) \ge s_k$$

tồn tại một nút x* trong một heap Fibonacci sao cho

$$s_k = \text{size}(x^*)$$

$$\ge 2 + \sum_{i=2}^{k} s_{degree[y_i]}$$



– vì s_k là tăng đơn điệu theo k, nên từ $degree[y_i] \ge i - 2$ (Lemma 21.1) ta có $s_{degree[y_i]} \ge s_{i-2}$. Vậy

$$s_k \ge 2 + \sum_{i=2}^{k} s_{i-2}$$

k1

Chặn trên lên bậc lớn nhất

Chứng minh (tiếp)

– dùng quy nạp theo k để chứng minh rằng $s_k \geq F_{k+2}$, với $k \geq 0$:

- Bước cơ bản: với $k = 0$ và $k = 1$ là rõ ràng.
- Bước quy nạp:
  – Giả thiết quy nạp: $k \geq 2$ và $s_i \geq F_{i+2}$ với $i = 0, 1, ..., k - 1$. Từ trên ta có

$$\begin{aligned} s_k &\geq 2 + \sum_{i=2}^{k} s_{i-2} \\ &\geq 2 + \sum_{i=2}^{k} F_i \\ &= 1 + \sum_{i=0}^{k} F_i \\ &= F_{k+2} \qquad \text{(Lemma 21.2)} \end{aligned}$$

– vậy: size(x) $\geq s_k \geq F_{k+2} \geq \phi^k$.

k1

vì
Fo=0
F1=1
khmt-hung, 12/8/2006

Chặn trên lên bậc lớn nhất (tiếp)

Hệ luận

Bậc lớn nhất $D(n)$ của nút bất kỳ trong một Fibonacci heap có n nút là $O(\lg n)$.

Chứng minh Dùng Lemma 21.3.

Proof Let x be any node in an n-node Fibonacci heap, and let k = *degree*[x].

By Lemma 21.3, we have $n >= size(x) >= \phi^{k}$

=> $k <= \log_{\phi}(n)$