Lecture 8: Các đống hợp nhất được

- Heap nhị phân
- Một *heap hợp nhất được* (mergeable heap) là một cấu trúc dữ liệu hỗ trợ năm thao tác sau (gọi là các *thao tác heap hợp nhất được*).
  - – MAKE-HEAP() tạo và trả về một heap trống.
  - – INSERT(H, x) chèn nút x, mà trường *key* của nó đã được điền, vào heap H .
  - – MINIMUM(H) trả về một con trỏ chỉ đến nút trong heap H mà khóa của nó là nhỏ nhất.
  - – EXTRACT-MIN(H) tách ra nút có khóa nhỏ nhất khỏi H, và trả về một con trỏ chỉ đến nút đó.
  - – UNION(H_1, H_2) tạo và trả về một heap mới chứa tất cả các nút của các heaps H_1 và H_2 . Các heaps H_1 và H_2 sẽ bị hủy bởi thao tác này.

Thời gian chạy của các thao tác lên heaps hợp nhất được

- n là số nút của heap

Thủ tục\Heap	Binary heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (khấu hao)
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	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$

Heap nhị thức

- Heap nhị thức

  Heap nhị thức được Vuillemin giới thiệu năm 1978.

  Hỗ trợ thêm các thao tác

  – DECREASE-KEY(H, x, k) gán vào nút x trong heap H trị mới k của khóa, k nhỏ hơn hay bằng trị hiện thời của khóa.

  – DELETE(H, x) xóa nút x khỏi heap H.

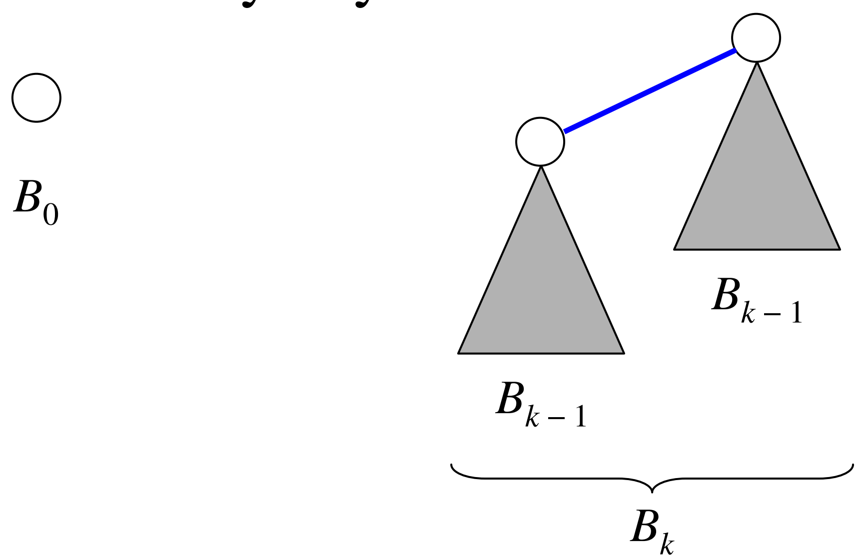
- Nhận xét:

  – Heap nhị thức không hỗ trợ thao tác SEARCH hữu hiệu.

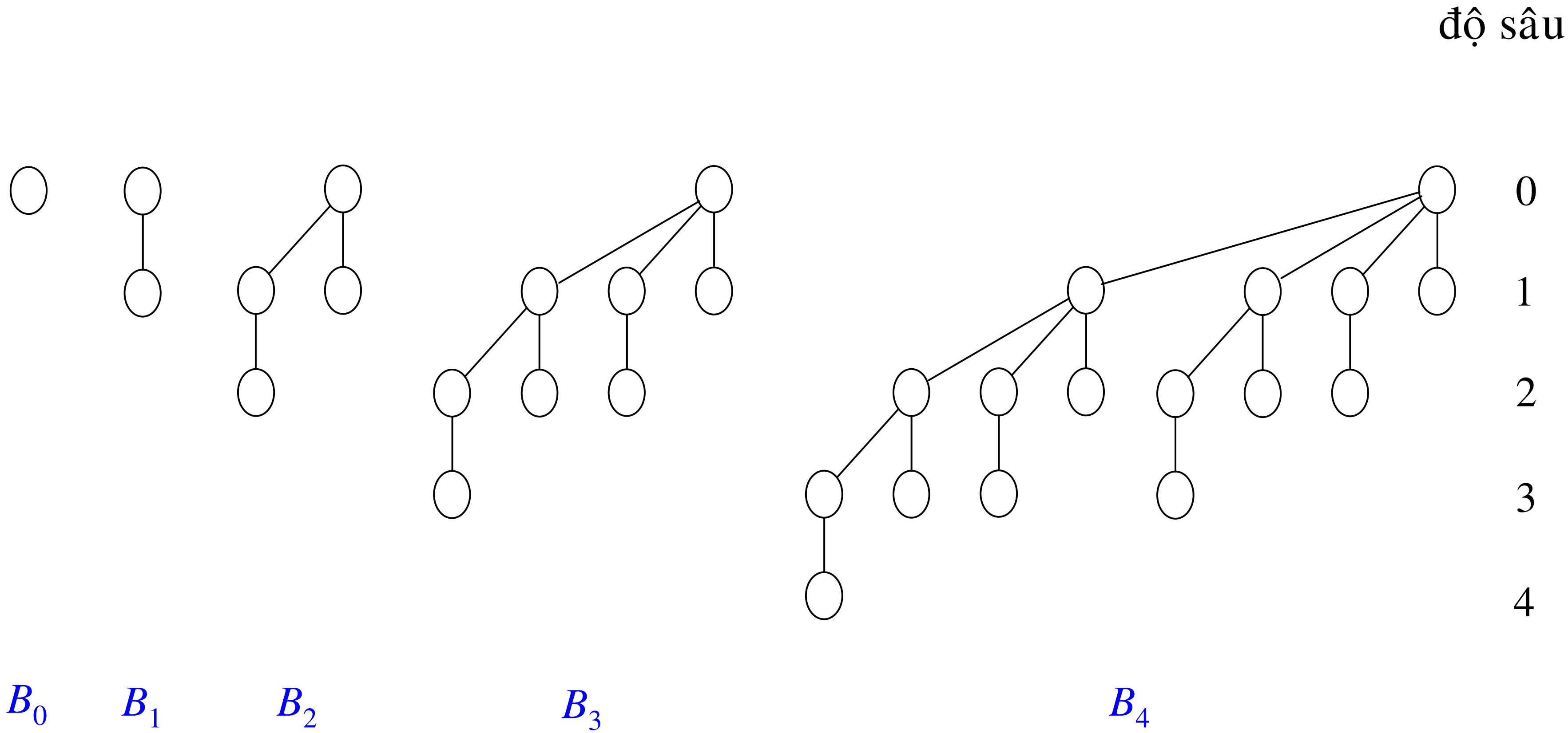
  – Do đó, các thao tác DECREASE-KEY và DELETE cần một con trỏ đến nút cần được xử lý.

Định nghĩa cây nhị thức

- *Cây nhị thức* B_k với $k = 0, 1, 2,...$ là một cây có thứ tự được định nghĩa đệ quy:
  - – Cây nhị thức B_0 gồm một nút duy nhất.
  - – Cây nhị thức B_k gồm hai cây nhị thức B_{k-1} được *liên kết* với nhau theo một cách nhất định:
    - Nút gốc của cây này là con bên trái nhất của nút gốc của cây kia.



Định nghĩa cây nhị thức



Đặc tính của cây nhị thức

- **Lemma – (**Đặc tính của một cây nhị thức**)**

  Cây nhị thức B_k có các tính chất sau:

  1. có 2^k nút,

  2. chiều cao của cây là k,

  3. có đúng $\binom{k}{i}$ nút tại độ sâu i với $i = 0, 1,..., k$

  4. bậc của nút gốc của cây là k, nó lớn hơn bậc của mọi nút khác; ngoài ra nếu các con của nút gốc được đánh số từ trái sang phải bằng $k - 1, k - 2,..., 0$, thì nút con i là gốc của cây con B_i .

$\binom{k}{i} = \frac{k!}{i!(k-i)!}$ *hệ số nhị thức (tổ hợp chập i từ k phần tử)*

Đặc tính của cây nhị thức

Chứng minh

Dùng quy nạp theo k.

Bước cơ bản: dễ dàng thấy các tính chất là đúng cho $B_0 - 2^0 = 1$ nút

Bước quy nạp: giả sử lemma là đúng cho B_{k-1} .

1. Cây nhị thức B_k gồm hai B_{k-1} nên B_k có $2^{k-1} + 2^{k-1} = 2^k$ nút.

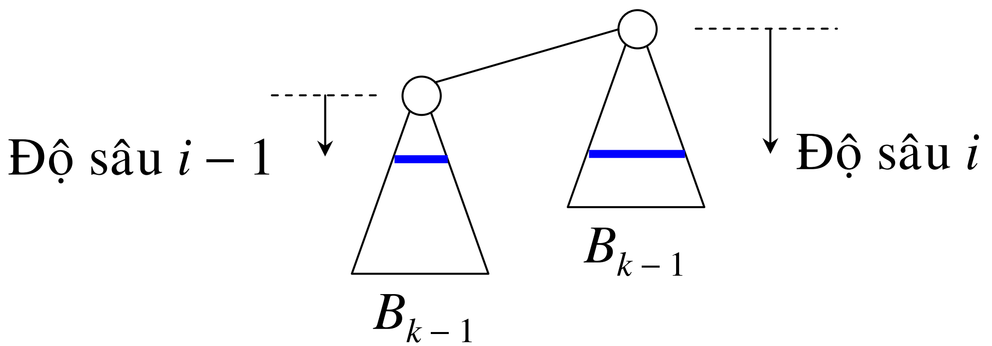
2. Do cách liên kết hai cây nhị thức B_{k-1} với nhau để tạo nên B_k nên độ sâu tối đa của nút trong B_k bằng độ sâu tối đa của nút trong B_{k-1} cộng thêm 1, tức là: $(k-1) + 1 = k$.

k1

Đặc tính của cây nhị thức

Chứng minh (tiếp)

3. Gọi $D(k, i)$ là số các nút tại độ sâu i của cây nhị thức B_k .



$$D(k,i) = D(k-1,i) + D(k-1,i-1)$$

$$= \binom{k-1}{i} + \binom{k-1}{i-1}$$

$$= \binom{k}{i}$$

k1

$D(k-1,i) = (k-1)!/(i!\,(k-i-1)!)$
$= (k!/k)\,(\,(k-i)/(i!(k-i)!)\,)$

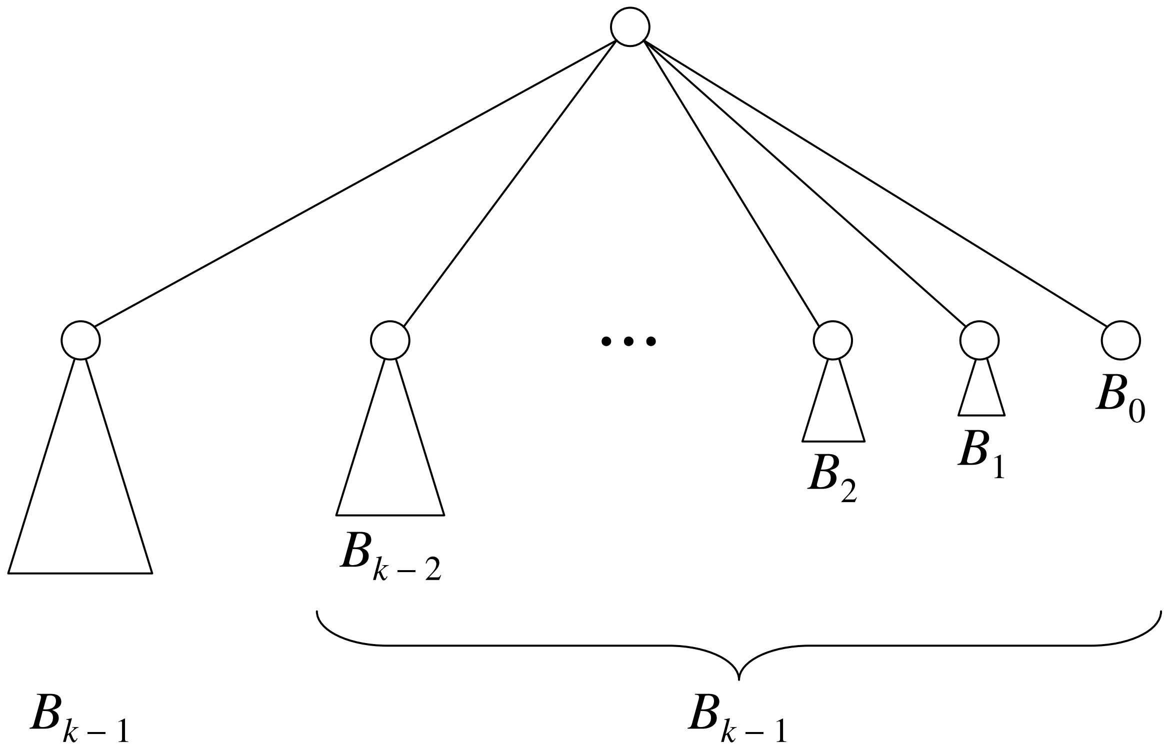
$D(k-1,i-1) = (k-1)!/(\,(i-1)!\,(k-i)!)$
$=(k!/k)\,(\,i/i!\,(k-i)!)$

khmt-hung, 11/29/2006

Đặc tính của cây nhị thức

Chứng minh (tiếp)

4. Sử dụng hình sau.Cây B_k gồm hai cây B_{k-1}



Đặc tính của cây nhị thức

- **Hệ quả (Corollary)**

  Bậc tối đa của một nút bất kỳ trong một cây nhị thức có *n* nút là lg2(*n)*.

Vì n=2^k -----– k là bậc của nút gốc (là bậc tối đa)

Định nghĩa heap nhị thức

- **Định nghĩa:** Một *heap nhị thức H* là một tập các cây nhị thức thỏa mãn các *tính chất heap nhị thức* sau:
  - 1. Mọi cây nhị thức trong *H* là *heap-ordered*: mọi nút đều có khóa lớn hơn hay bằng khóa của nút cha của nó.
  - 2. Với mọi số nguyên $k \geq 0$ cho trước thì có nhiều nhất một cây nhị thức trong *H* mà gốc của nó có bậc là *k*.

Tính chất của heap nhị thức

- **Tính chất**

  1. Gốc của một cây trong một heap nhị thức chứa khóa nhỏ nhất trong cây.
  2. Một heap nhị thức H với n nút gồm nhiều nhất là $\lfloor \lg 2(n) \rfloor + 1$ cây nhị thức.

Chứng minh

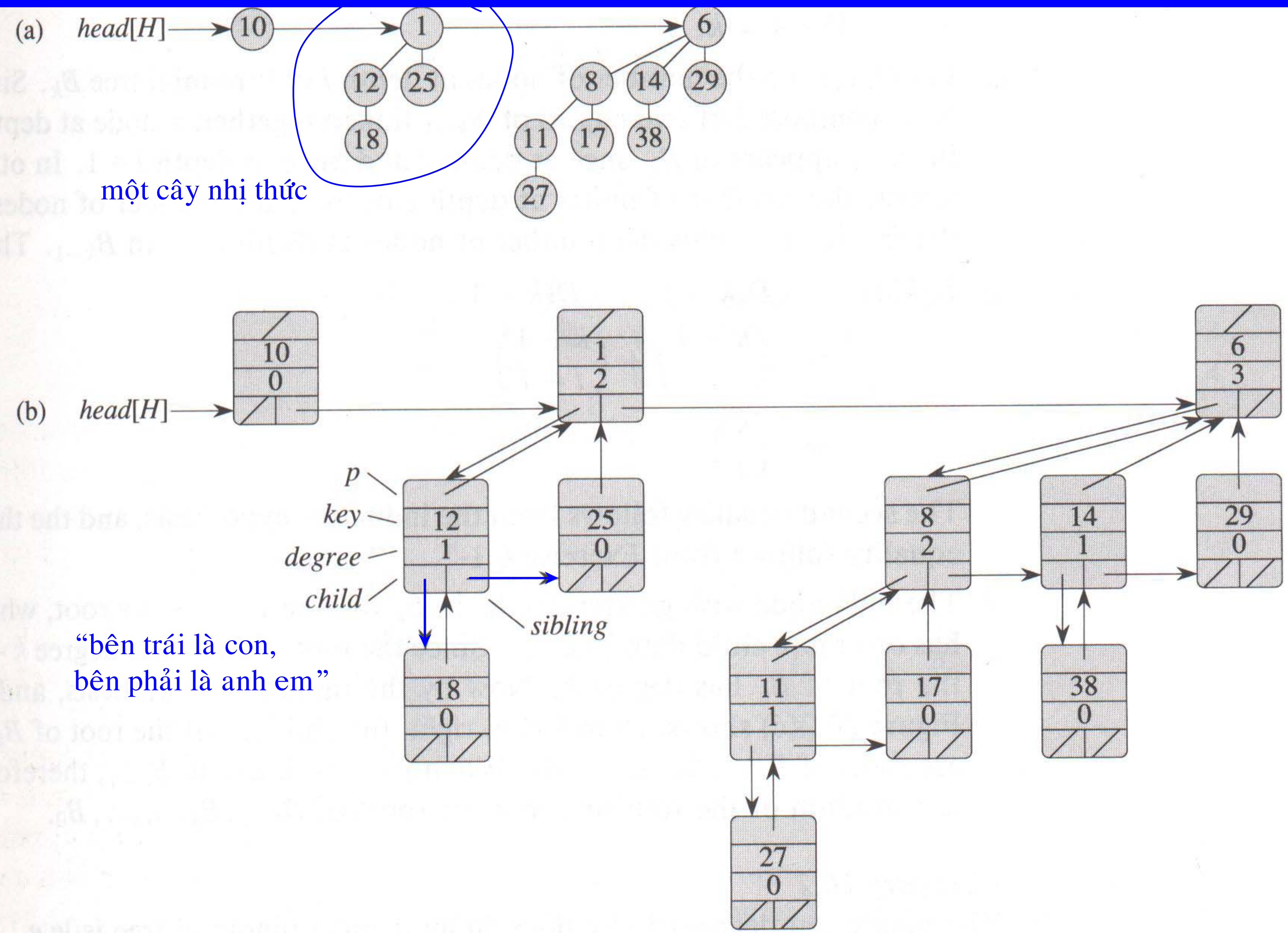
1. Hiển nhiên.
2. n có biểu diễn nhị phân duy nhất, biểu diễn này cần $\lfloor \lg n \rfloor + 1$ bits, có dạng $\langle b_{\lfloor \lg n \rfloor}, b_{\lfloor \lg n \rfloor - 1}, \ldots, b_0 \rangle$ sao cho

$$n = \sum_{i=0}^{\lfloor \lg n \rfloor} b_i 2^i$$

$$10 = \begin{array}{cccc} 3 & 2 & 1 & 0 \\ \hline \multicolumn{1}{|c|}{1} & \multicolumn{1}{c|}{0} & \multicolumn{1}{c|}{1} & \multicolumn{1}{c|}{0} \\ \hline \end{array}$$

Cùng với định nghĩa 2, ta thấy cây nhị thức *Bi* xuất hiện trong *H* nếu và chỉ nếu *bi* = 1. => Với n=10 thì trong heap có 2 cây nhị thức là B1 và B3

Biểu diễn heap nhị thức



Biểu diễn heap nhị thức

Qui tắc trữ cho mỗi cây nhị thức trong một heap nhị thức:

- biểu diễn theo kiểu "*Bên trái là con, bên phải là anh em*" (left-child, right-sibling representation)

- Mỗi nút x có một trường sau
  - *key*[x]: trữ khóa của nút.

  và các con trỏ sau:
  - *p*[x]: trữ con trỏ đến nút cha của x.
  - *child*[x]: con trỏ đến con bên trái nhất của x.
    - Nếu x không có con thì *child*[x] = NIL
  - *sibling*[x]: con trỏ đến anh em của x ở ngay bên phải x.
    - Nếu x là con bên phải nhất của cha của nó thì *sibling*[x] = NIL.

Biểu diễn heap nhị thức (tiếp)

- Ngoài ra mỗi nút *x* còn có một trường sau
  - *degree*[*x*]: bậc của *x* (= số các con của *x*)
- Các gốc của các cây nhị thức trong một heap nhị thức được tổ chức thành một danh sách liên kết, gọi là *danh sách các gốc* của heap nhị thức.
  - Khi duyệt danh sách các gốc của một heap nhị thức thì các bậc của các gốc theo thứ tự tăng dần.
  - Nếu *x* là một gốc thì *sibling*[*x*] chỉ đến gốc kế đến trong danh sách các gốc.
- Để truy cập một heap nhị thức *H*
  - *head*[*H*]: con trỏ chỉ đến gốc đầu tiên trong danh sách các gốc của *H*.
    - *head*[*H*] = NIL nếu *H* không có phần tử nào.

Tạo một heap nhị thức

- Thủ tục để tạo một heap nhị thức mới:

  MAKE-BINOMIAL-HEAP

  – Tạo một đối tượng H với $head[H]$ = NIL.

  – có thời gian chạy là $\Theta(1)$.

k3

Tìm khóa nhỏ nhất

- Thủ tục để tìm khóa nhỏ nhất trong một heap nhị thức *H* có *n* nút: BINOMIAL-HEAP-MINIMUM
  - trả về một con trỏ đến nút có khóa nhỏ nhất.

```
BINOMIAL-HEAP-MINIMUM(H)
y ← NIL
x ← head[H]
min ← ∞
while x ≠ NIL
      do if key[x] < min
            then min ← key[x]
                 y ← x
         x ← sibling[x]
return y
```

  - Thời gian chạy của thủ tục là $O(\lg n)$ vì cần kiểm tra nhiều nhất là $\lfloor \lg n \rfloor + 1$ nút gốc.

Slide 17

k3 Nếu có n nút thì suy ra có tối đa là lg2(n) + 1 cây nhị thức
khmt-hung, 11/29/2006

Liên kết hai cây nhị thức

- Thủ tục để liên kết hai cây nhị thức:

  BINOMIAL-LINK

  – liên kết cây nhị thức B_{k-1} có gốc tại nút y vào cây nhị thức B_{k-1} có gốc tại nút z để tạo ra cây nhị thức B_k . Nút z trở thành gốc của một cây B_k .

```
BINOMIAL-LINK(y, z)
   p[y] ← z
   sibling[y] ← child[z]
   child[z] ← y
   degree[z] ← degree[z] + 1
```

  – Thời gian chạy của thủ tục là $O(1)$.

Hòa nhập hai heap nhị thức

- Thủ tục để hòa nhập (merge) danh sách các gốc của heap nhị thức H_1 và danh sách các gốc của heap nhị thức H_2 :

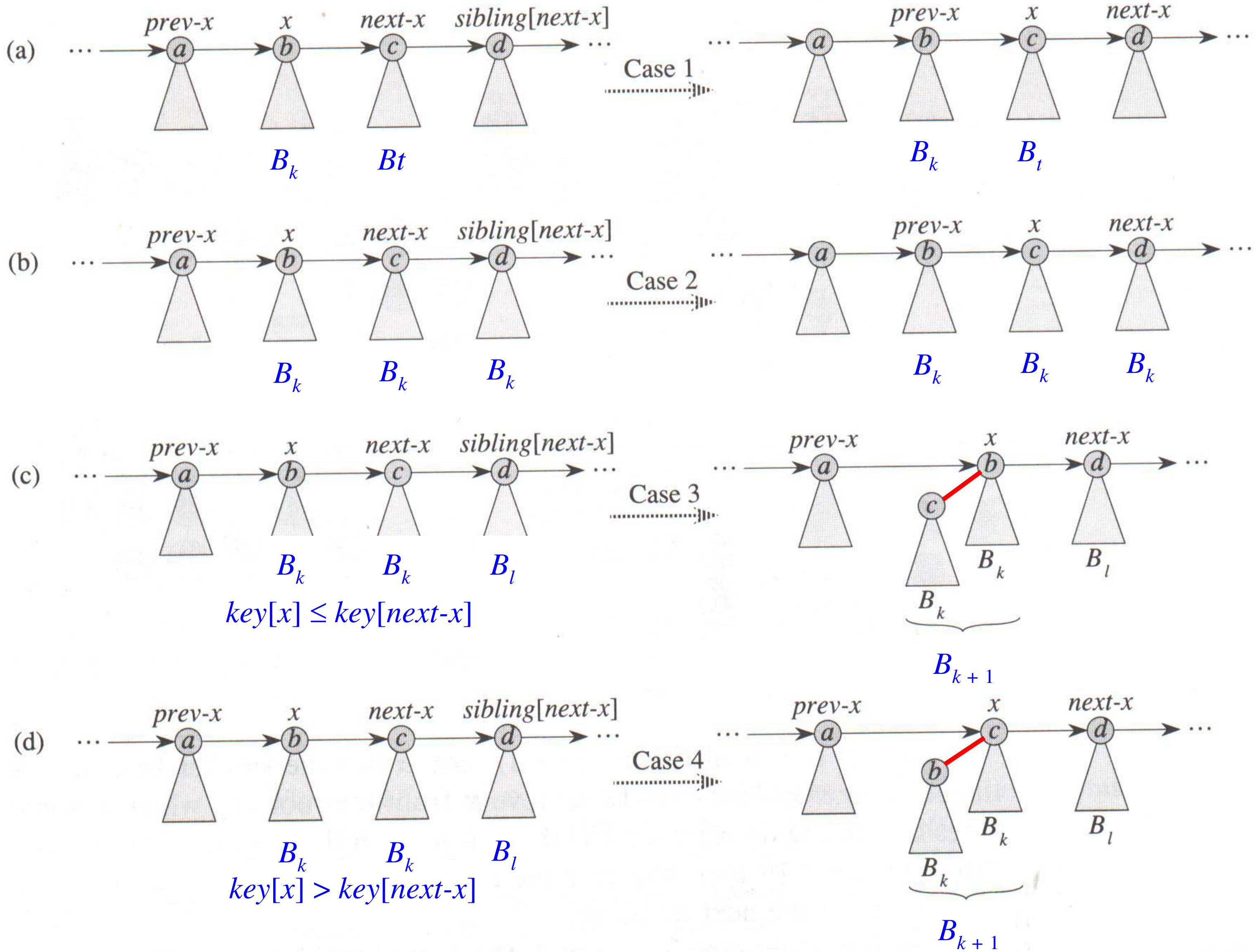
  BINOMIAL-HEAP-MERGE(H_1 , H_2)

  – hòa nhập các danh sách các gốc của H_1 và H_2 thành một danh sách các gốc duy nhất mà thứ tự các bậc là tăng dần.

  – nếu các danh sách các gốc của H_1 và H_2 có tổng cộng là m gốc, thì thời gian chạy của thủ tục là $O(m)$.

k4

Các trường hợp xảy ra trong BINOMIAL-HEAP-UNION



k4 các cây Bk và Bt (t>k)
khmt-hung, 11/29/2006

Hợp hai heap nhị thức

- Thủ tục để hợp hai heap nhị thức:

  BINOMIAL-HEAP-UNION

  – hợp nhất hai heap nhị thức H_1 và H_2 và trả về heap kết quả.

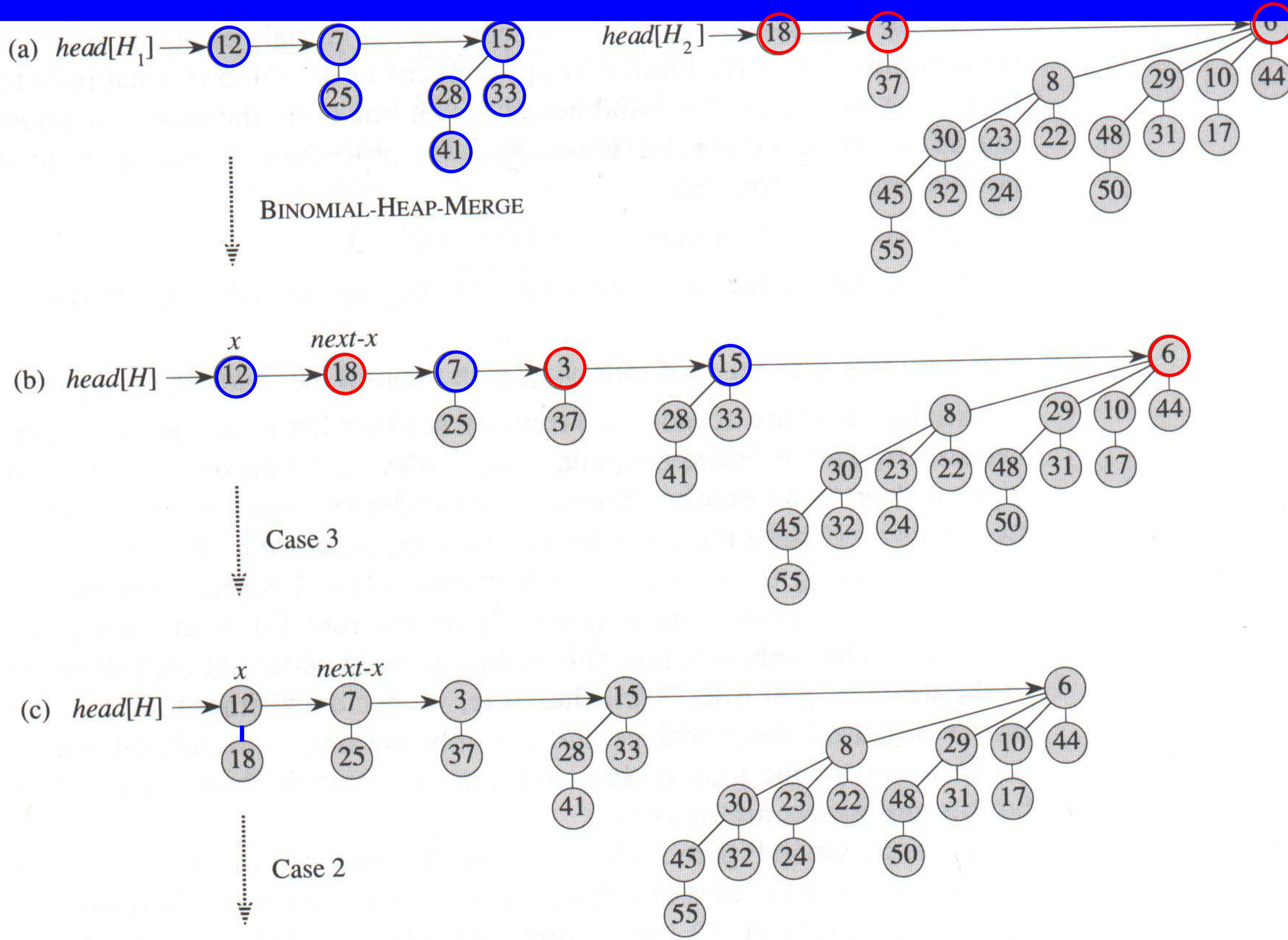
```
BINOMIAL-HEAP-UNION(H1 , H2)
    H ← MAKE-BINOMIAL-HEAP()
    head[H] ← BINOMIAL-HEAP-MERGE(H1 , H2)
    // Giải phóng H1 và H2
    if head[H] = NIL
       then return H
    prev-x ← NIL
    x ← head[H]
    next-x ← sibling[x]
```

Hợp hai heap nhị thức

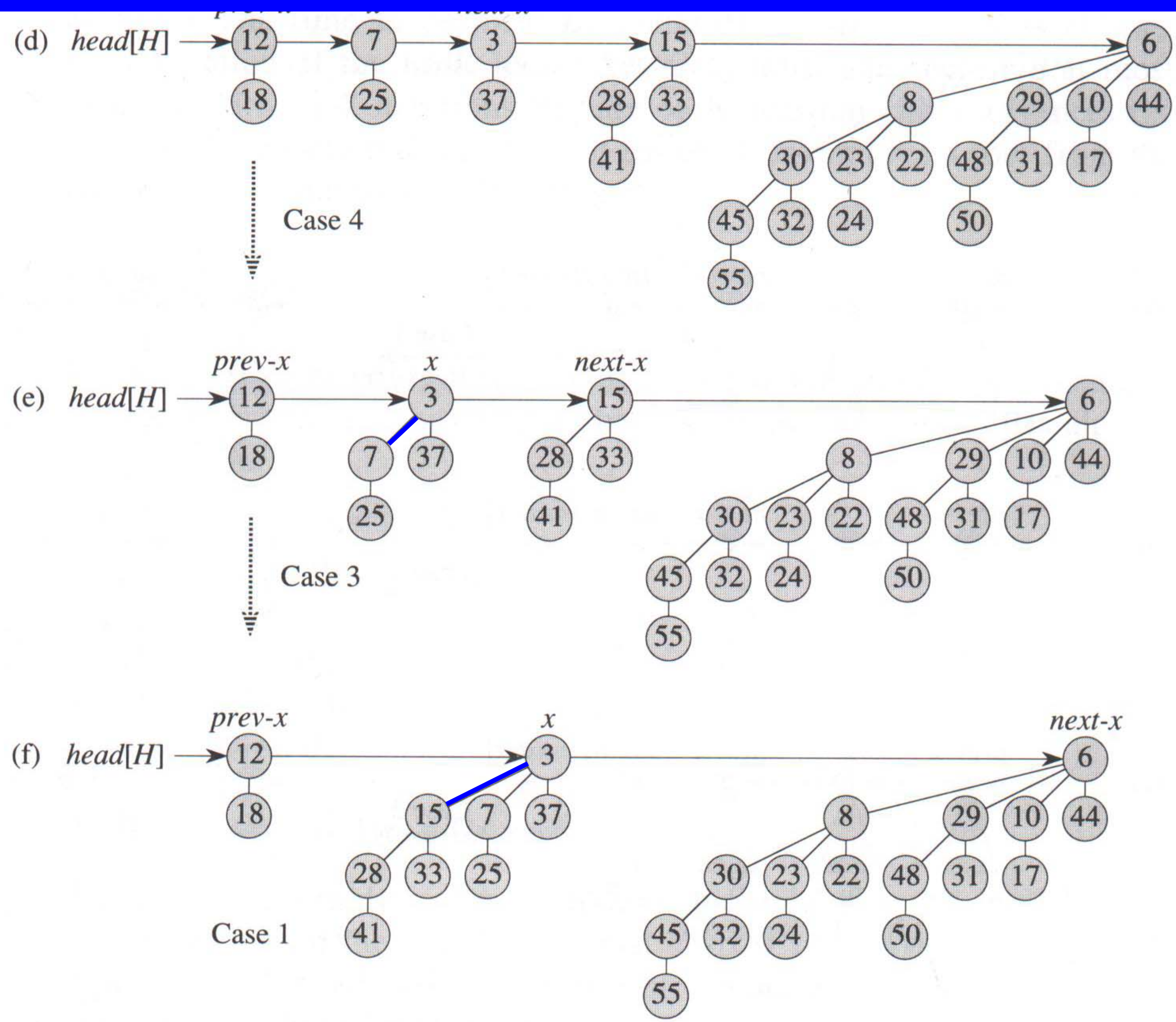
(tiếp)

```
while next-x ≠ NIL
    do if (degree[x] ≠ degree[next-x] hay
                (sibling[next-x] ≠ NIL
                 và degree[sibling[next-x]] = degree[x])
          then prev-x ← x                              ▷ Trường hợp 1 và 2
               x ← next-x                              ▷ Trường hợp 1 và 2
          else if key[x] ≤ key [next-x]
                  then sibling[x] ← sibling[next-x]    ▷ Trường hợp 3
                       BINOMIAL-LINK(next-x, x)        ▷ Trường hợp 3
                  else if prev-x = NIL                 ▷ Trường hợp 4
                          then head[H] ← next-x        ▷ Trường hợp 4
                          else sibling[prev-x] ← next-x ▷ Trường hợp 4
                       BINOMIAL-LINK(x, next-x)        ▷ Trường hợp 4
                       x ← next-x                      ▷ Trường hợp 4
       next-x ← sibling[x]
return H
```

Ví dụ thực thi BINOMIAL-HEAP-UNION



Ví dụ thực thi BINOMIAL-HEAP-UNION (tiếp)



Phân tích BINOMIAL-HEAP-UNION

- Thời gian chạy của BINOMIAL-HEAP-UNION là $O(\lg n)$, với n là số nút tổng cộng trong các heaps H_1 và H_2 . Đó là vì
  - Gọi n_1 là số nút của H_1, và n_2 là số nút của H_2 , ta có $n = n_1 + n_2$.
  - Do đó H_1 chứa tối đa $\lfloor \lg n_1 \rfloor + 1$ nút gốc, và H_2 chứa tối đa $\lfloor \lg n_2 \rfloor + 1$ nút gốc. Vậy BINOMIAL-HEAP-MERGE chạy trong thời gian $O(\lg n)$.
  - H chứa tối đa $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor + 2 = O(\lg n)$ nút ngay sau khi thực thi xong BINOMIAL-HEAP-MERGE. Do đó vòng lặp **while** lặp tối đa $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor + 2$ lần, mỗi lần lặp tốn $O(1)$ thời gian.

Chèn một nút

- Thủ tục để chèn một nút vào một heap nhị thức:

  BINOMIAL-HEAP-INSERT

  – chèn một nút *x* vào một heap nhị thức *H*, giả sử đã dành chỗ cho *x* và khóa của *x*, *key*[*x*], đã được điền vào.

```
BINOMIAL-HEAP-INSERT(H, x)
1   H' ← MAKE-BINOMIAL-HEAP()
2   p[x] ← NIL
3   child[x] ← NIL
4   sibling[x] ← NIL
5   degree[x] ← 0
6   head[H'] ← x
7   H ← BINOMIAL-HEAP-UNION(H, H')
```

  – Thời gian chạy của thủ tục là $O(\lg n)$.

Tách ra nút có khóa nhỏ nhất

- Thủ tục để tách ra nút có khóa nhỏ nhất khỏi heap nhị thức:
  BINOMIAL-HEAP-EXTRACT-MIN
  – đem nút có khóa nhỏ nhất khỏi heap nhị thức *H* và trả về một con trỏ chỉ đến nút được tách ra.

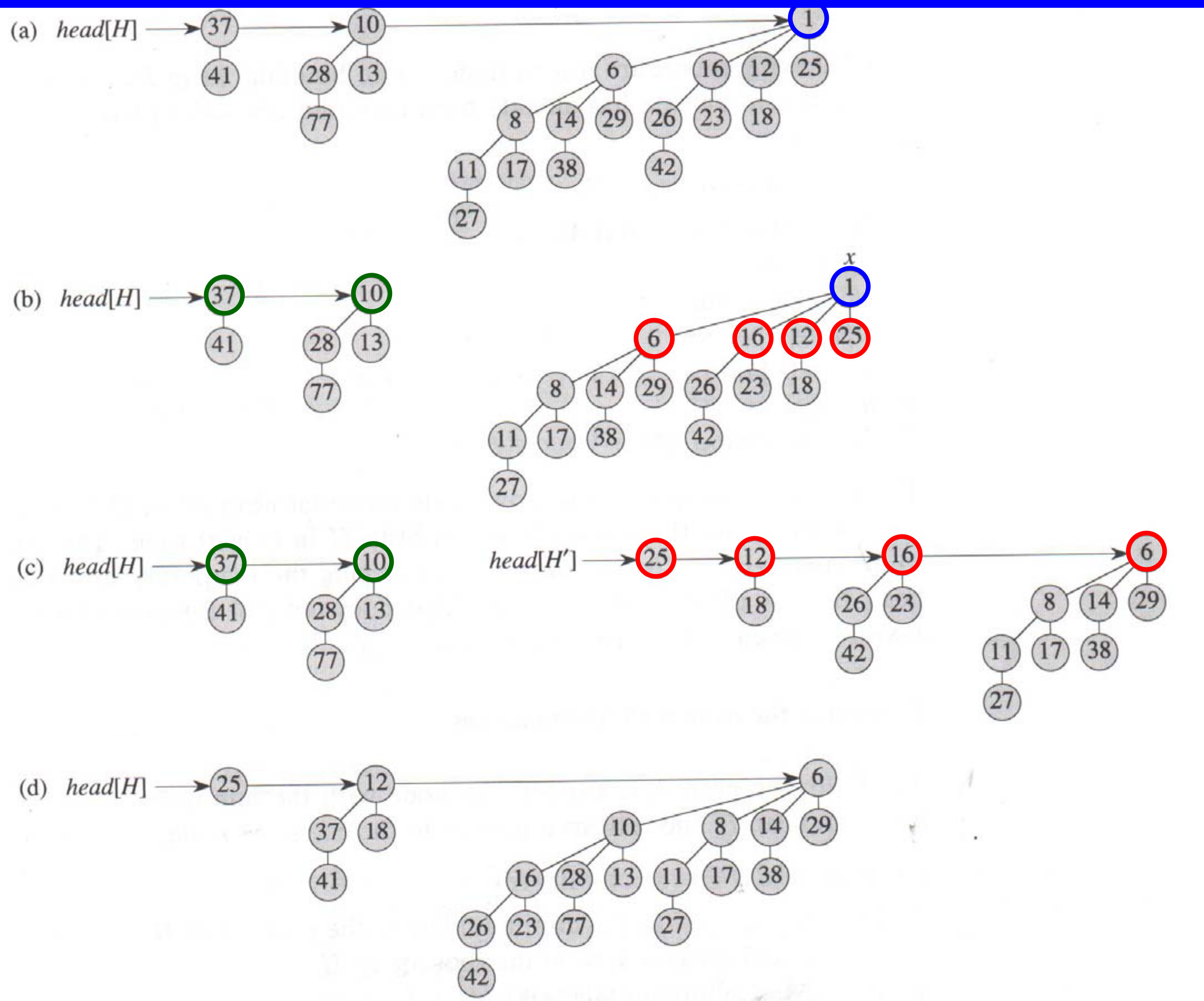
BINOMIAL-HEAP-EXTRACT-MIN(*H*)
tìm trong danh sách các gốc của *H* gốc *x* có khóa nhỏ nhất, và đem *x* ra khỏi danh sách các gốc của *H*
H' ← MAKE-BINOMIAL-HEAP()
đảo ngược thứ tự của các con của *x* trong danh sách liên kết của chúng, và gán vào *head*[*H'*] con trỏ chỉ đến đầu của danh sách có được
H ← BINOMIAL-HEAP-UNION(*H*, *H'*)
return *x*

Tách ra nút có khóa nhỏ nhất

(tiếp)

– Thời gian chạy của thủ tục là $O(\lg n)$ vì nếu H có n nút thì mỗi dòng từ 1 đến 4 thực thi trong thời gian $O(\lg n)$.

Ví dụ thực thi BINOMIAL-HEAP-EXTRACT-MIN



Giảm khóa

- Thủ tục để giảm khóa của một nút trong một heap nhị thức thành một trị mới:

  BINOMIAL-HEAP-DECREASE-KEY

  – giảm khóa của một nút *x* trong một heap nhị thức *H* thành một trị mới *k*.

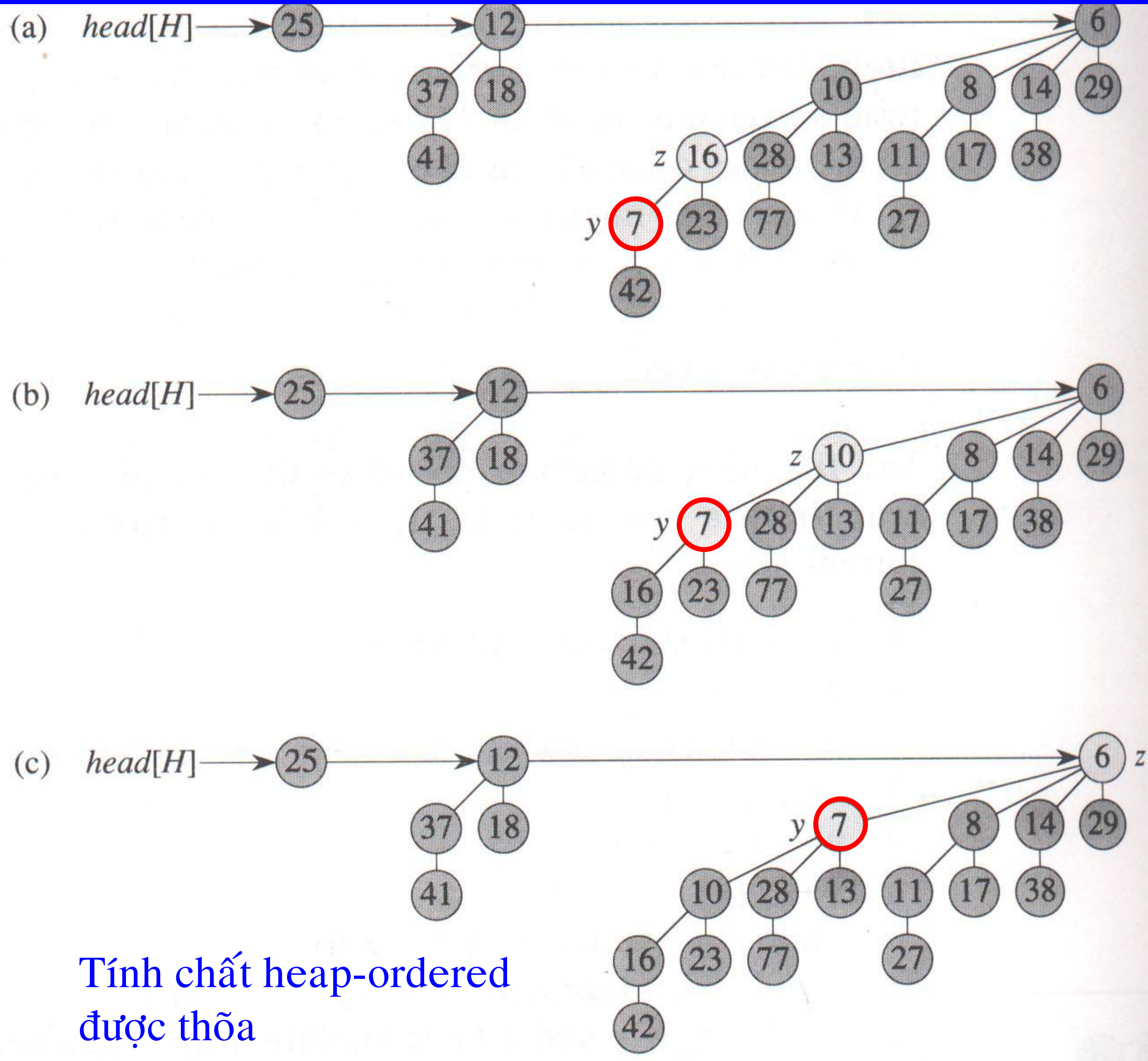
Giảm khóa

– Tính chất *heap-ordered* của cây chứa x phải được duy trì!

```
BINOMIAL-HEAP-DECREASE-KEY(H, x, k)
 1  if k > key[x]
 2      then error "khóa mới lớn hơn khóa hiện thời"
 3  key[x] ← k
 4  y ← x
 5  z ← p[y]
 6  while z ≠ NIL và key[y] < key[z]
 7          do đổi chỗ key[y] ↔ key[z]
 8              ▷ Nếu y và z có thông tin phụ thì cũng đổi chỗ chúng
 9              y ← z
10              z ← p[y]
```

– Thời gian chạy của thủ tục là $O(\lg n)$: vì x có độ sâu tối đa là $\lfloor \lg n \rfloor$ nên vòng lặp **while** (dòng 6-10) lặp tối đa $\lfloor \lg n \rfloor$ lần.

Ví dụ thực thi BINOMIAL-HEAP-DECREASE-KEY



Tính chất heap-ordered
được thõa

Xóa một khóa

- Thủ tục để xóa khóa của một nút *x*:

  BINOMIAL-HEAP-DELETE

  – xóa khóa của một nút *x* khỏi heap nhị thức *H*.

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

  – Thời gian chạy của thủ tục là $O(\lg n)$.

Bài toán cây khung nhỏ nhất

1. Cho G=(V,E) là đồ thị vô hướng liên thông với tập đỉnh V={1,2,…,n} và tập cạnh E gồm m cạnh. Mỗi cạnh được gán một giá trị thực c(e) – độ dài của cạnh.
2. Bài toán đặt ra là tìm cây khung có độ dài nhỏ nhất.
3. Thuật toán Kruskal: Ban đầu cho T=rỗng, tìm cạnh nhỏ nhất trong E ghép cạnh này vào T nếu không tạo ra chu trình. Đến khi trong T có n-1 cạnh thì T chính là cây khung tối thiểu.
4. Thuật toán Prim: (phương pháp lân cận gần nhất). Xuất phát từ một đỉnh tùy ý s, y là một đỉnh lân cận gần nhất của s. Tiếp tục như thế với 2 đỉnh s và y ta lại tìm một đỉnh z lân cận gần nhất của 2 đỉnh này … Tiếp tục cho đến khi ta thu được n đỉnh.

THUẬT TOÁN TÌM CÂY KHUNG NHỎ NHẤT SỬ DỤNG ĐỐNG NHỊ THỨC

Giả sử có một đồ thị vô hướng liên thông $G=(V,E)$ với hàm trọng số $w:E \rightarrow R$.

Ta gọi $w(u,v)$ là trọng số của cạnh (u,v). Ta muốn tìm một khung nhỏ nhất với G: một tập hợp con các cây khung $T \subseteq E$ liên thông tất cả các đỉnh trong V có tổng trọng số là nhỏ nhất.

Thuật toán tìm cây khung tối thiểu T:

Ta gọi $\{Vi\}$ là một phân hoạch các đỉnh của V và với mỗi tập Vi, ta có một tập hợp $Ei \subseteq \{(u,v): u \in V$ và $v \in Vi$.

MST-MERGEABLE-HEAP(G)

T ← ∅
For mỗi đỉnh $vi \in$ V[G] **do**
    $Vi \leftarrow \{vi\}$
    $Ei \leftarrow \{(vi,v) \in$ E[G]$\}$
While còn có nhiều hơn một tập đỉnh Vi **do**
    chọn bất kỳ tập đỉnh Vi
    trích cạnh có trọng số cực tiểu (u,v) từ Ei
    với $u \in Vi$ và $v \in Vj$
    If $i \neq j$ **then**
        $T \leftarrow T \cup \{(u,v)\}$
        $Vi \leftarrow Vi \cup Vj$, huỷ Vj
        $Ei \leftarrow Ei \cup Ej$

Khai báo cấu trúc

```
struct        BinomialEdge
    { int u,v;  // đỉnh u, v
     int key;  //  {trọng số cạnh w(u,v)}
     int degree; // số nút con
    struct BinomialEdge* child;
    struct BinomialEdge* sibling;
    struct BinomialEdge* parent;
    }
// Danh sách các đống nhị thức
BinomialEdge* E [N];
```

Thuật toán chi tiết trên cấu trúc đống nhị thức

```
MST-MERGEABLE-HEAP(G)
T ← ∅
For  mỗi đỉnh vi ∈ V[G]            do
        Vi ← {vi}
        Ei ← Make-Binomial-Heap()
        For      (vi,v) ∈ E[G] do
                 Binomial-Heap-Insert((vi,v), Ei)

While  có hơn một tập đỉnh Vi do
        chọn bất kỳ tập đỉnh Vi ≠ ∅
        (u,v) ← Binomial-Heap-Extract-Min(Ei)
        với u ∈ Vi và v ∈ Vj   {tìm tập Vj}
        If i ≠ j then
                 T ← T ∪ {(u,v)}
                 Vi ← Vi ∪ Vj, huỷ Vj
                 Ei ← Binomial-Heap-Union(Ei,Ej)
```

Bài tập

1. Thực hiện từng bước thuật toán trên với một bài toán cụ thể.
2. Cài đặt thuật tóan Prim với Binomial Heap trên (C++; C#, Java)