CHIẾN LƯỢC BIẾN ĐỔI ĐỂ TRỊ

- Các đặc trưng cơ bản
- Các ví dụ minh họa

CÁC ĐẶC TRƯNG CƠ BẢN

- Chiến lược biến đổi để trị (transform-and-conquer) gồm hai tầng (giai đoạn):
  - Biến đổi thể hiện (instance-input) của bài toán để có thể dễ giải (hoặc giải hiệu quả) hơn
  - Giải bài toán trên thể hiện (input) đã được biến đổi

CÁC ĐẶC TRƯNG CƠ BẢN

- Có ba kiểu biến đổi
  - Biến đổi về thể hiện đơn giản hoặc thuận tiện hơn của cùng bài toán, gọi là (instance simplification)
  - Biến đổi về một dạng biểu diễn khác của cùng thể hiện bài toán, gọi là (representation change)
  - Biến đổi về một thể hiện của một bài toán khác đã có giải thuật, gọi là (problem reduction)

CÁC VÍ DỤ MINH HỌA

- Bài toán kiểm tra tính duy nhất của một phần tử
- Giải thuật HeapSort
- Qui tắc Horner
- Tính tổng $S=1^3+2^3+...+n^3$
- Tính bội số chung nhỏ nhất

BÀI TOÁN KIỂM TRA TÍNH DUY NHẤT CỦA MỘT PHẦN TỬ

- Cho một danh sách các phần tử, kiểm tra các phần tử trong danh sách có duy nhất hay không (có phần tử nào xuất hiện ít nhất là hai lần hay không)

BÀI TOÁN KIỂM TRA TÍNH DUY NHẤT CỦA MỘT PHẦN TỬ

- Nếu dùng chiến lược trực tiếp sẽ có giải thuật $O(n^2)$
- Biến đổi để trị bằng cách sắp xếp (presorting) danh sách tăng dần (biến đổi đầu vào của bài toán) sau đó thực hiện việc kiểm tra tính duy nhất trên danh sách được sắp
- Giải thuật thực hiện chỉ một vòng lặp

BÀI TOÁN KIỂM TRA TÍNH DUY NHẤT CỦA MỘT PHẦN TỬ

ALGORITHM PresortElementUniqueness(A[0..n – 1])

1 Sort the array A

2 **for** i ←0 **to** n – 2 **do**

3 **if** A[i]= A[i + 1] **return** false

4 **return** true

BÀI TOÁN KIỂM TRA TÍNH DUY NHẤT CỦA MỘT PHẦN TỬ

- Kích thước đầu vào là n
- Thời gian sắp xếp là $\Theta(n\log_2 n)$ khi dùng giải thuật tốt (QuickSort)
- Thời gian kiểm tra dòng 2-3 không quá $\Theta(n)$

  $T(n)= \Theta(n\log_2 n)+ \Theta(n)= \Theta(n\log_2 n)$
- Lưu ý: Có nhiều bài toán áp dụng "tiền sắp xếp" (presorting) như một kỹ thuật biến đổi để trị

GIẢI THUẬT HEAPSORT

- Giải thuật HeapSort sắp xếp một mảng số tăng dần dùng chiến lược biến đổi để trị bằng cách biểu diễn lại các phần tử của mảng (biến đổi biểu diễn đầu vào) trong quá trình sắp
- Việc biến đổi được thực hiện nhờ cấu trúc dữ liệu HEAP

GIẢI THUẬT HEAPSORT

- Cho một mảng A, một Heap biểu diễn A là một cây nhị phân có các tính chất sau:
  - Cây có thứ tự, mỗi nút tương ứng với một phần tử của mảng, gốc ứng với phần tử đầu tiên của mảng
  - Cây cân bằng và được lấp đầy trên tất cả các mức, ngoại trừ mức thấp nhất được lấp đầy từ bên trái sang (có thể chưa lấp đầy)

GIẢI THUẬT HEAPSORT

- Một MaxHeap là một Heap mà giá trị của mỗi nút lớn hơn (hoặc bằng) giá trị các nút con

GIẢI THUẬT HEAPSORT

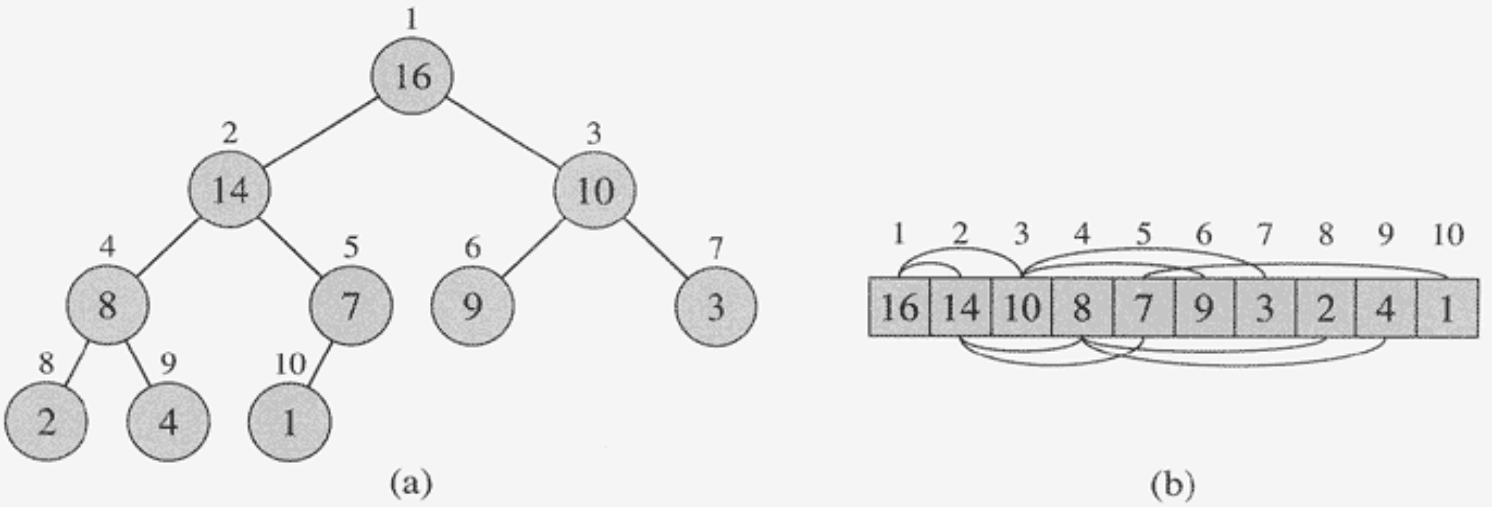

Figure 6.1 A max-heap viewed as **(a)** a binary tree and **(b)** an array. The number within the circle at each node in the tree is the value stored at that node. The number above a node is the corresponding index in the array. Above and below the array are lines showing parent-child relationships; parents are always to the left of their children. The tree has height three; the node at index 4 (with value 8) has height one.

GIẢI THUẬT HEAPSORT

- Gọi i là chỉ số thứ tự của một nút (của mảng), thì chỉ số các nút cha, con trái và con phải là:
  - PARENT(i) = $\lfloor i/2 \rfloor$
  - LEFT_CHILD(i) = $2i$
  - RIGHT_CHILD(i)= $2i+1$

GIẢI THUẬT HEAPSORT

- Giải thuật HeapSort biến đổi mảng về MaxHeap để sắp xếp như sau:
  - Biến đổi Heap (mảng) về một MaxHeap
  - Hoán đổi giá trị A[1] với A[n]
  - Loại nút n ra khỏi Heap và chuyển Heap A[1..(n-1)] thành một MaxHeap (ít hơn một phần tử)
  - Lặp lại các bước trên cho đến khi Heap chỉ còn một phần tử

GIẢI THUẬT HEAPSORT

- Cần hai thủ tục (giải thuật) hỗ trợ cho HeapSort:
  - MaxHeappify(A[1..n], i), biến đổi cây con có gốc tại i thành một MaxHeap (gốc tại i)
  - BuildMaxHeap(A[1..n]), biến đổi mảng (Heap) thành MaxHeap

MaxHeapify

- Đầu vào là một mảng (heap) A và chỉ số i trong mảng
- Các cây nhị phân được định gốc tại LEFT_CHILD(i) và RIGHT_CHILD(i) là các MaxHeap nhưng A[i] có thể nhỏ hơn các con của nó
- MaxHeapify đẩy giá trị A[i] xuống sao cho cây con định gốc tại A[i] là một MaxHeap

MaxHeapify

```
MaxHeapify(A[1..n],  i)
1     l ← 2*i; r ← 2*i+1
2     if l<=n and A[l]>A[i]  largest ← l
3     else largest ← i
4     if r<=n and A[r]>A[largest]  largest ← r
5     if largest ≠ i
6           Exchange(A[i], A[largest])
7           MaxHeapify(A, largest)
```

BuildMaxHeap

- Các nút có chỉ số $\lfloor n/2 \rfloor + 1$, $\lfloor n/2 \rfloor + 2$, ..., n trong A[1..n] là các lá của cây, mỗi nút như vậy là một MaxHeap
- BuildMaxHeap áp dụng MaxHeapify cho các nút con khác lá của cây từ dưới lên gốc bắt đầu từ nút $\lfloor n/2 \rfloor$
- Kết quả là một MaxHeap tương ứng với A[1..n]

BuildMaxHeap

```
BuildMaxHeap(A[1..n])
1   for i ← n/2 downto 1
2       do  MaxHeapify(A, i)
```

BuildMaxHeap

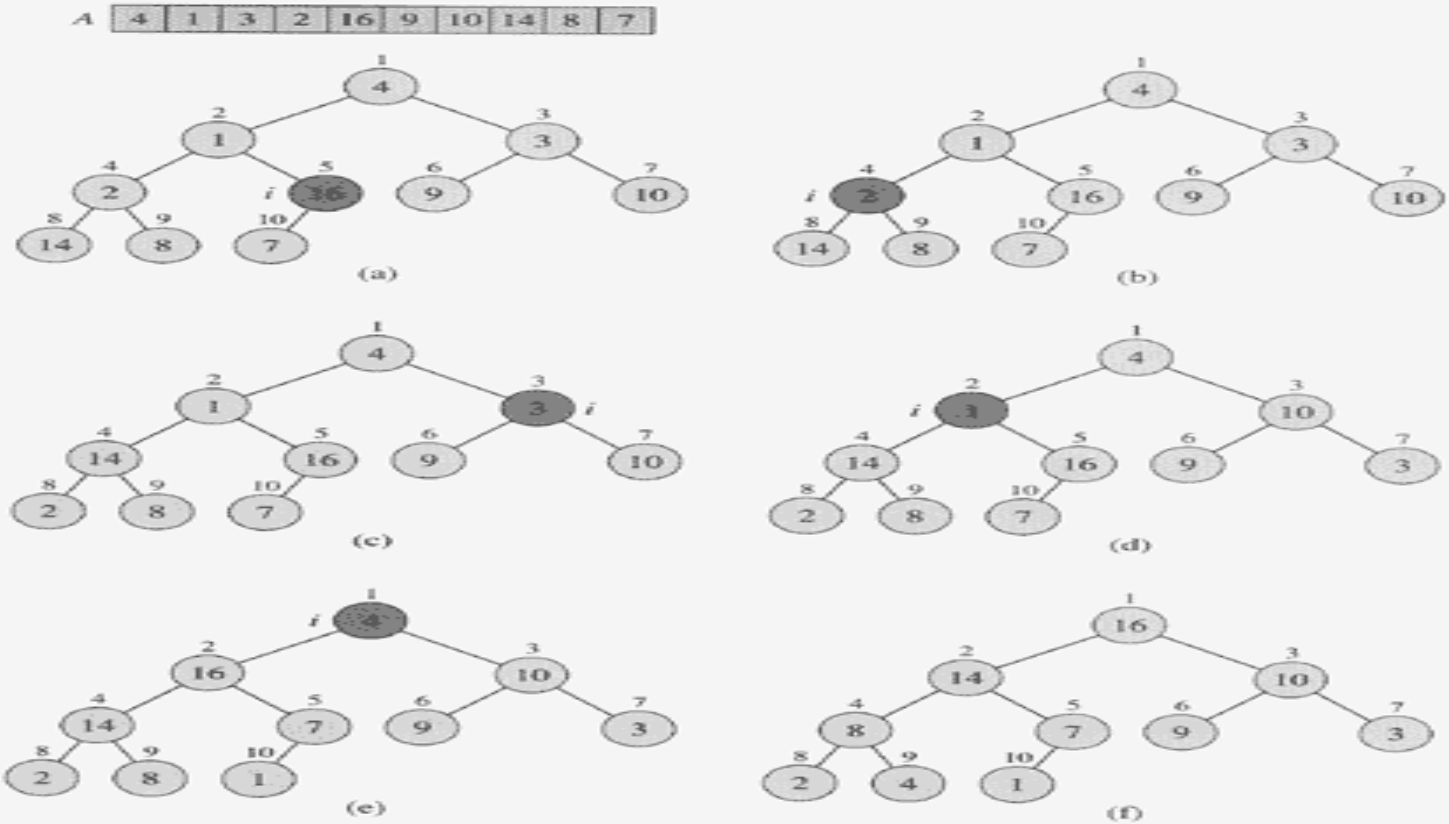

Figure 6.3 The operation of BUILD-MAX-HEAP, showing the data structure before the call to MAX-HEAPIFY in line 3 of BUILD-MAX-HEAP. **(a)** A 10-element input array A and the binary tree it represents. The figure shows that the loop index i refers to node 5 before the call MAX-HEAPIFY(A, i). **(b)** The data structure that results. The loop index i for the next iteration refers to node 4. **(c)–(e)** Subsequent iterations of the **for** loop in BUILD-MAX-HEAP. Observe that whenever MAX-HEAPIFY is called on a node, the two subtrees of that node are both max-heaps. **(f)** The max-heap after BUILD-MAX-HEAP finishes.

HeapSort

```
HeapSort(A[1..n])
1   BuildMaxHeap(A)
2   for i ← n downto 2
3       do  Exchange(A[1], A[i])
4           MaxHeapify(A, i-1, 1)  // MaxHeapify(A[1..i-1], 1)
```

HeapSort

- Kích thước đầu vào là n (số phần tử của mảng)
- Gọi chiều cao của cây là h, do cây cân bằng nên $h = \lfloor \log_2 n \rfloor$
  - Thời gian chạy của MaxHeapify là $O(h)=O(\log_2 n)$
  - Thời gian chạy của BuilMaxHeap tối đa là $O(n\log_2 n)$
  - Thời gian chạy của vòng lặp 2-4 là $O(n\log_2 n)$
- Vậy thời chạy của HeapSort là

$$T(n)= O(n\log_2 n)+O(n \log_2 n)=O(n\log_2 n)$$

QUI TẮC HORNER

- Cho đa thức $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$, tính giá trị của đa thức tại x cho trước

QUI TẮC HORNER

- Biến đổi đa thức $p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$ về một dạng (biểu diễn) mới

$$p(x) = \underbrace{x(x \ldots x(x}_{\text{n-1 thừa số}}(a_n x + a_{n-1}) + a_{n-2}) + \ldots + a_1) + a_0$$

➢ Chuyển bài toán về tính tích các nhị thức

QUI TẮC HORNER

$$p(x) = 2x^4 - x^3 + 3x^2 + x - 5$$

$$= x(2x^3 - x^2 + 3x + 1) - 5$$

$$= x(x(2x^2 - x + 3) + 1) - 5$$

$$= x(x(x(2x - 1) + 3) + 1) - 5$$

QUI TẮC HORNER

ALGORITHM Horner(a[0..n], x)

//a[0], a[1], .., a[n] là các hệ số tương ứng với x^0, x^1, …, x^n

1 p ← a[n]

2 **for** i ←n – 1 **downto** 0

3 **do** p←x ∗ p + a[i]

4 **return** p

QUI TẮC HORNER

- Kích thước đầu vào là n
- Thao tác cơ bản là phép toán nhân
- Thời gian thực hiện thao tác là c

$$T(n)=nc=\Theta(n)$$

TÍNH TỔNG $S=1^3+2^3+\ldots+n^3$

- Biến đổi bài toán tính tổng về bài toán tính biểu thức (bài toán mới)

$$S=1^3+2^3+\ldots+n^3=n^2(n+1)^2/4$$

TÍNH TỔNG $S=1^3+2^3+\ldots+n^3$

SumPowerThree(n)

1 **return** n*n *(n+1)*(n+1) /4

Độ phức tạp T(n)=O(1)

Lưu ý: Giải thuật trực tiếp là O(n)

TÍNH BỘI SỐ CHUNG NHỎ NHẤT

- Tính bội số chung nhỏ nhất của hai số m, n cho trước
  - Ta đã biết giải thuật, gcd(m, n), tính ước số chung lớn nhất (greatest common divisor) của hai số m, n
  - Biến đổi bài toán tính bội số chung nhỏ nhất (least common multiple) về bài toán tính ước số chung lớn nhất

    lcm(m, n)=n.m/gcd(m, n)

TÍNH BỘI SỐ CHUNG NHỎ NHẤT

ALGORITHM lcm(m, n)

1 **return** n*m/gcd(m, n)

TÍNH BỘI SỐ CHUNG NHỎ NHẤT

ALGORITHM gcd (m, n)

//Input: Two nonnegative, not-both-zero integers m and n

//Output: Greatest common divisor of m and n

1 **while** n ≠ 0 **do**

2 r ←m mod n

3 m←n

4 n←r

5 **return** m

TÍNH BỘI SỐ CHUNG NHỎ NHẤT

- Giả sử max(m, n)≥8, khi đó giải thuật gcd (m, n) có độ phức tạp là $O(\log_{3/2}(2.\max(m, n)/3) \approx O(\log_2 n)$
- Từ đó giải thuật lcm(m, n) cũng có độ phức tạp

$$O(\log_{3/2}(2.\max(m, n)/3) \approx O(\log_2 n)$$

BÀI TẬP VỀ NHÀ

- Đọc chương 6 (Transform-and-Conquer), sách Levitin
- Làm bài tập về nhà đã cho trong DS bài tập
- Bài thực Hành: Hiện thực HeapSort so sánh với MergeSort và SelectionSort