Lecture 6: Heaps

Pure Consciousness is a Field

Perfect Order and Balance

1

Wholeness Statement

A heap is a binary tree that stores sortable elements at each internal node and maintains the *heap-order* property and is *complete* (balanced). Heap-order means that for every node *v* (except the root), *key*(*v*)≥*key*(*parent*(*v*)). *Science of Consciousness*: Pure consciousness is the field of wholeness, perfectly orderly, balanced, and complete. Through regular TM practice we release stress and automatically develop the qualities of the unified field in our lives.

2

The Heap Data Structure

3

Min-Heap ADT

- A Heap stores a collection of sortable elements
- Main methods of the Heap ADT
  - insertElem(e) inserts and returns the new Position (node) inserted into the Heap that contains the element e
  - removeMin() removes and returns the smallest element in the Heap
- Additional methods
  - minElem() returns the smallest element, but does not remove it
  - size(), isEmpty()

4

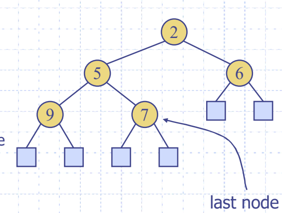
Comparator

- A comparator encapsulates the action of comparing two objects according to a given total order relation
- A generic Heap uses an auxiliary comparator
- The comparator is external to the elements being compared
- When the Heap needs to compare two elements, it uses its comparator
- The Comparator Method returns a number as its return type indicating the relative size of its input arguments
  - compare(x, y) returns a positive number if x<y, zero if x=y, and a negative number if x>y

5

What is a heap

- A heap is a binary tree storing keys at its internal nodes and satisfying the following properties:
  - Heap-Order: for every internal node v other than the root, $key(v) \geq key(parent(v))$
  - Complete Binary Tree: let h be the height of the heap
    - for $i = 0, \ldots, h-1$, there are 2^i nodes of depth i
    - at depth $h-1$, the internal nodes are to the left of the external nodes
- The last node of a heap is the rightmost internal node of depth $h-1$



6

Heap-Order Property

- For all internal nodes v (except the root):

  $key(v) \geq key(parent(v))$

  - That is, the key of every child node is greater than or equal to the key of its parent node

7

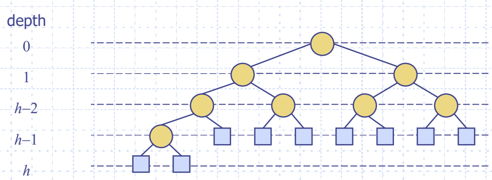
Other Properties of a Heap

- A heap is a binary tree whose values are in ascending order on every path from root to leaf
- Values are stored in internal nodes only
- A heap is a binary tree whose root contains the minimum value and whose subtrees are heaps

8

Heap

- All leaves of the tree are on two adjacent levels
- The binary tree is complete on every level except the deepest level.



9

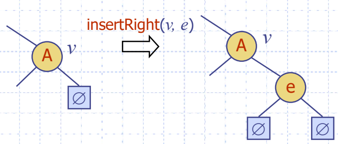
Adding Nodes to a Heap

- New nodes must be added left to right at the lowest level, i.e., the level containing internal and external nodes or containing all external nodes

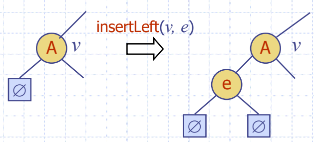
10

Binary Tree Insert Methods

insertRight(*v, e*)
Right child must be external!



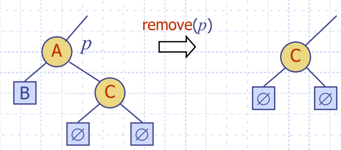
insertLeft(*v, e*)
Left child must be external.

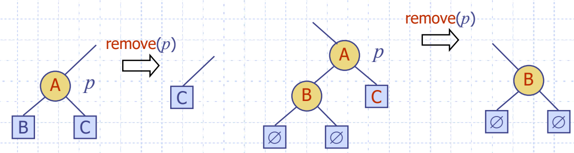


11

Binary Tree Remove Method

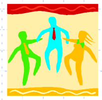
remove(*p*)
Either the left or right child must be external! We can only remove the node above an external node.



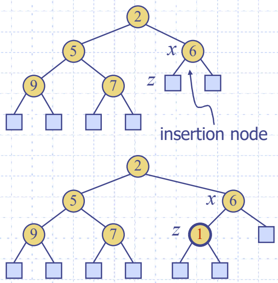


12

Insertion into a Heap



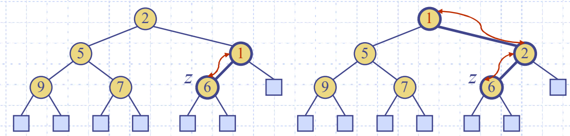
- Important property of a min-heap
  - The smallest element is at the root
- The insertElem(k) insertion algorithm consists of three steps
  1. Find the insertion node x (above new last node z)
  2. Store k below x and expand z into an internal node (either insertRight(x,k) or insertLeft(x,k))
  3. Restore the heap-order property (Upheap)



13

Upheap

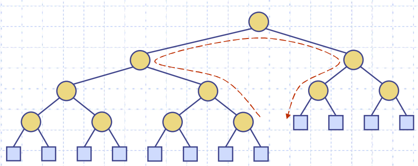
- After the insertion of a new key k, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping k along an upward path from the insertion node
- Upheap terminates when the key k reaches the root or a node whose parent has a key smaller than or equal to k
- Since a heap has height $O(\log n)$, upheap runs in $O(\log n)$ time



14

Finding the Insertion Node and Updating the Last Node

- The insertion node can be found by traversing a path of $O(\log n)$ nodes
  - While the current node is a right child, go to the parent node
  - If the current node is a left child, go to the right child
  - While the current node is internal, go to the left child
- Similar algorithm for updating the last node after a removal



15

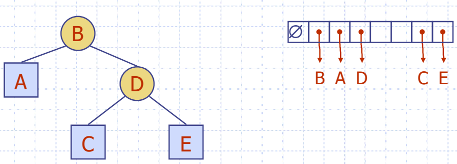
Efficient Representation of A Heap

Use an Array or Sequence with efficient random access

16

Efficient Data Structure for Implementing the Heap

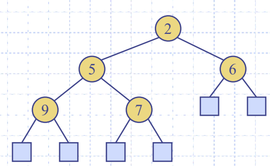
- Heap implementation: use an array to store the binary tree.
- Node objects are referenced by index:
  - Index 0 is empty and not used.
  - Root node is at index 1
  - Left child is at 2*index
  - Right child is at 2*index+1
  - Position would contain the element and index into the array



17

Sequence (or Array) based Heap Implementation

- We can represent a heap with n keys by means of a vector of length $n + 1$
- For the node at rank i
  - the left child is at rank $2i$
  - the right child is at rank $2i + 1$
- Links between nodes are not explicitly stored
- The leaves are not represented
- The cell at rank 0 is not used
- Operation *insertElem* corresponds to inserting at rank $n + 1$
- Operation *removeMin* corresponds to removing at rank n
- No empty cells in the array
- Yields in-place heap-sort



	2	5	6	9	7
0	1	2	3	4	5

18

Exercise

- Insert the following keys into a heap represented as an array-based Tree:
  9, 6, 5, 14, 4, 12, 15, 3, 2

19

Array-Based Implementation of Binary Tree

Operation	Time
size, isEmpty	
elements	
swapElements(p, q), replaceElement(p, e)	
root(), parent(p), children(p)	
_isInternal(p), _isExternal(p), _isRoot(p)	

20

Array-Based Implementation of Binary Tree

Operation	Time
size, isEmpty	1
positions, elements	n
swapElements(p, q), replaceElement(p, e)	1
root(), parent(p), children(p)	1
isInternal(p), isExternal(p), isRoot(p)	1

21

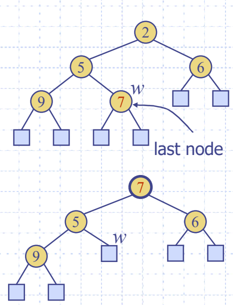
Pseudo Code for upHeap

```
Algorithm upHeap(H, size)
   Input Array H representing a heap and the size of H (size ≥ 1)
   Output H with the heap property restored

   i ← size
   parent ← floor(size/2)
   while 1 ≤ parent ∧ key(H[parent]) > key(H[i]) do
         swapElements(H[parent], H[i])
         i ← parent
         parent ← floor(i/2)

Algorithm swapElements(p, q)
     temp ← p.element
     p.element ← q.element          {swap elements}
     q.element ← temp
```


22

Removal from a Heap

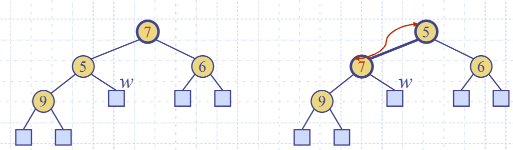
- Method removeMin of the priority queue ADT corresponds to the removal of the root key from the heap
- The removal algorithm consists of three steps
  - Replace the root key with the key of the last node w
  - Compress w and its children into a leaf
  - Restore the heap-order property



23

Downheap

- After replacing the root key with the key k of the last node, the heap-order property may be violated
- Algorithm downheap restores the heap-order property by swapping key k along a downward path from the root
- Downheap terminates when key k reaches a leaf or a node whose children have keys greater than or equal to k
- Since a heap has height $O(\log n)$, downheap runs in $O(\log n)$ time



24

Iterative Version of downHeap

```
Algorithm downHeap(H, size)
    Input Array H representing a heap and the size of H (size ≥ 1)
    Output H with the heap property restored

    property ← false
    i ← 1
    while ! property do
        smallest ← indexOfMin(H, i, size)
        if smallest ≠ i then
            swapElements(H[smallest], H[i])
            i ← smallest
        else
            property ← true

    return H
```


25

Helper for downHeap Algorithm

```
Algorithm indexOfMin(A, r, size)
    Input array A, an index r (referencing an item of A), and size of the
          heap stored in A
    Output the rank of element in A containing the smallest value

    smallest ← r
    left ← 2*r
    right ← 2*r + 1
    if left ≤ size ∧ key(A[left]) < key(A[smallest]) then
        smallest ← left
    if right ≤ size ∧ key(A[right]) < key(A[smallest]) then
        smallest ← right
    return smallest
```


26

Analysis of Heap Operations

- Upheap()
- Downheap()
- insertElem(element)
- removeMin()
  - removes element with minimum key if a min-heap (need a comparator for items)
  - removeMax() removes element with maximum key if a Max-Heap (parent is greater or equal to its children)
- The Heap data structure is the basis of the HeapSort and is the basis of the proper implementation of a PriorityQueue (covered tomorrow)

27

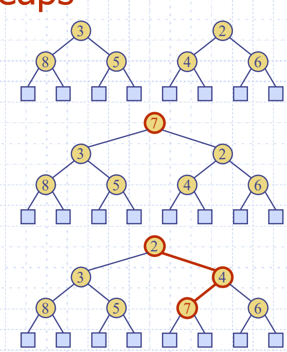
Main Point

1. A heap is a binary tree that stores sortable elements at each internal node and maintains *heap-order* and is *complete*. Heap-order means that for every node *v* (except the root), *key*(*v*)≥*key*(*parent*(*v*)). *Science of Consciousess*: Pure consciousness is the field of wholeness, perfectly orderly, and complete. Through regular TM practice we release stress and automatically develop the qualities of the unified field in our lives.

28

Merging Two Heaps

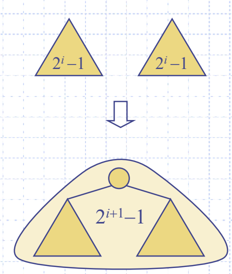
- We are given two heaps and a key k
- We create a new heap with the root node storing k and with the two heaps as subtrees
- We call downHeap to restore the heap-order property



29

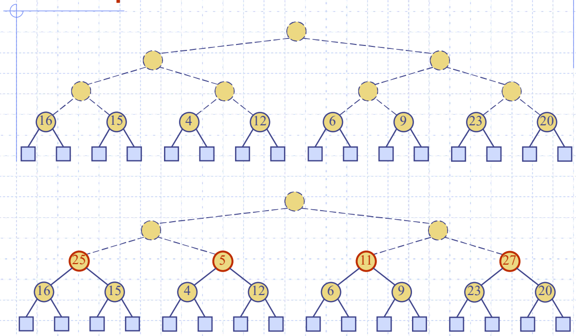
Bottom-up Heap Construction

- We can construct a heap storing n given keys using a bottom-up construction with $\log n$ phases
- In phase i, pairs of heaps with $2^i - 1$ keys are merged into heaps with $2^{i+1} - 1$ keys



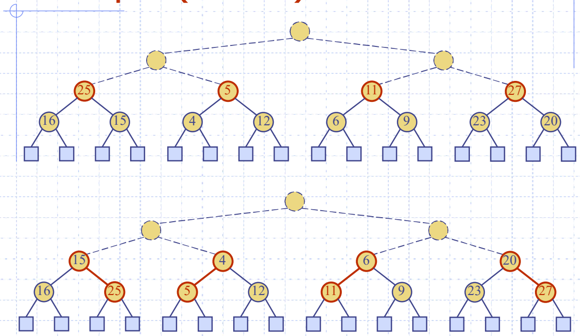
30

Example



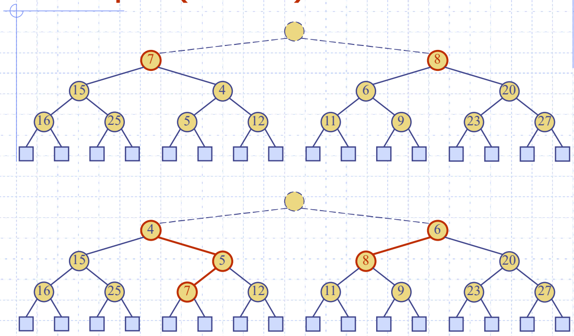
31

Example (contd.)



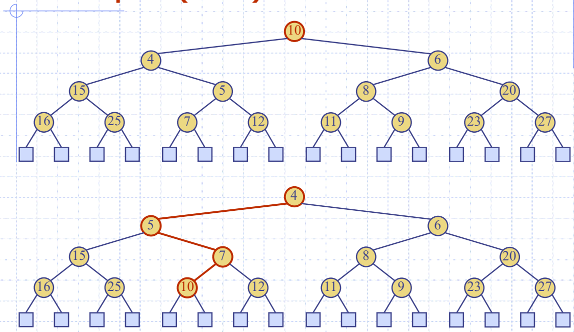
32

Example (contd.)



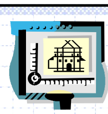
33

Example (end)

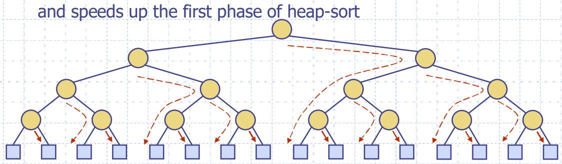


34

Analysis of Bottom-up Heap Construction



- We visualize the worst-case time of a downheap with a proxy path that goes first right and then repeatedly goes left until the bottom of the heap (this path may differ from the actual downheap path)
- Since each node is traversed by at most two proxy paths, the total number of nodes of the proxy paths is $O(n)$
- Thus, bottom-up heap construction runs in $O(n)$ time
- Bottom-up heap construction is faster than n successive insertions and speeds up the first phase of heap-sort



35

Connecting the Parts of Knowledge with the Wholeness of Knowledge

1. The Tree ADT models a hierarchical structure between objects simplified to a parent-child relation. A Heap is a binary tree with two properties, each path from root to leaf is in sorted order and the tree is always balanced.
2. The Heap can be implemented in two ways and thus its operations will have different algorithms, e.g., the binary tree can be implemented as either a set of recursively defined nodes or as an array of elements.

36

3. **Transcendental Consciousness** is pure intelligence, the abstract substance out of which the universe is made.
4. **Impulses within Transcendental Consciousness**: Within this field, the laws of nature continuously organize and govern all activities and processes in creation.
5. **Wholeness moving within itself** : In Unity Consciousness, awareness is awake to its own value, the full value of the intelligence of nature. One's consciousness supports the knowledge that outer is the expression of inner, creation is the play and display of the Self.