CMPT 225
Binary Heaps
Last Lectures

- Tree Sort
Unit Learning Outcomes

At the end of this unit, a student will be able to:

- Abstract Data Types (ADTs):
  - define binary heaps
    and demonstrate, trace and implement their operations
  - given a set of requirements, select the most appropriate data collection ADT class (and its most appropriate underlying data structure), taking into account its time and space efficiency
    - Binary heap may be selected as an ADT or as a data structure
- Sorting:
  - implement and analyze heap sort
- Recursion:
  - write recursive definitions of binary heap
Today’s Menu

- Our goal is to
  - define binary heaps,
  - demonstrate, trace their recursive operations
Array-based implementations of Binary Tree

- There is another flavour of Binary Tree that can be efficiently implemented using an array-based implementation
Binary Heap

Definition:

A complete binary tree can be implemented efficiently using an array since there are no gaps in such tree.
Review – Complete Binary Tree
Maximum Binary Heap

- Maximum Binary Heap is a …
  - complete binary tree, and
  - the key value of a node in such heap is > or = to key value of its children (if any), and
  - the node’s left and right subtrees are also maximum binary heaps

- In such a heap, the root contains the element with the largest key value
Example

Maximum Binary Heap

Example:

Complete?
Max Heap?

Root - d largest key value.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5</td>
<td>7</td>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Minimum Binary Heap

- Minimum Binary Heap is a …
  - complete binary tree, and
  - the key value of a node in such heap is \(<\) or \(=\) to key value of its children (if any), and
  - the node’s left and right subtrees are also minimum binary heaps

- In such a heap, the root contains the element with the smallest key value
Example

Minimum Binary Heap

Example:

Root \to Smallest key value.

3 4 5 4 7 8
Operations for a data collection

Heap ADT

- Insert
- Delete
  - Delete the root
- Retrieve
  - Behaves like Min( ) or Max( ) of BST
- Number of elements?
Public Interface for data collection
Heap ADT

(UML syntax)

// Description: Inserts an element
+ insert ( element: ElementType ) : void

// Description: Deletes the element located at the root and returns it.
// Precondition: Heap is not empty.
// Throws EmptyHeapException when heap is empty.
+ delete( ) : ElementType

// Description: Returns the element located at the root
// Minimum or maximum
// Precondition: Heap is not empty.
// Postcondition: The heap is unchanged by this operation.
// Throws EmptyHeapException when heap is empty.
+ retrieve( ) : ElementType

// Description: Returns the number of elements currently stored in the data collection.
// Postcondition: The heap is unchanged by this operation.
+ getElementCount( ) : integer

Note that we can also implement delete such that it does not return the deleted element.

This decision is made by the Heap ADT class designer, based on the problem to be solved.
Array-based Implementation of a Heap – Review

- Considering cell at index i
  - its left child/subtree is located at index: $2 \times i + 1$
  - its right child/subtree is located at index: $2 \times i + 2$
  - its parent is located at index: $\text{floor}( (i – 1) / 2 )$

- However, ensure that the computed indices are $<\text{CAPACITY}$ of array, i.e., indices within bound of array

This is the most common implementation of a heap
A Note About the Tracing of Binary Heap Algorithm Execution

- In our lecture notes, all binary heap algorithms (insert, delete and sort) manipulate an array since a binary heap is implemented using an array as its underlying data structure.
- This is the reason why we often draw an array as we are tracing the execution of the algorithm.
A Note About the Tracing of Binary Heap Algorithm Execution

- However, as we are tracing the execution of the algorithm, we also often draw the (conceptual) tree representation of the binary heap.
- The reason we do this is because it is often easier to visualize how the algorithm executes when the binary heap is represented as a tree.
- However, understand that the algorithm does not create the tree representation of the binary heap, hence it does not require additional memory space for the tree.
Insertion into a Min Binary Heap

Algorithm:
indexOfRoot = 0
indexOfBottom = elementCount
insert new element @ “bottom” of heap (@ indexOfBottom)
elementCount ++
reHeapUp( indexOfBottom )

// If the element has a parent and …
if ( indexOfBottom > indexOfRoot )
    indexOfParent = floor((indexOfBottom - 1) / 2)
    // … key value of parent > key value of child then …
    if ( heap[indexOfParent] > heap[indexOfBottom] )
        //… swap the element with its parent
        swap heap[indexOfParent] with heap[indexOfBottom]
        reHeapUp( indexOfParent )
Insertion – Let’s Try!

Insert ____ in Min binary heap:
Insertion – Let’s Try!

Insert ____ in Min binary heap:

```
3
/  \
3   4
/    /
5    7
```

```
Insertion – Let’s Try!

Insert _____ in Min binary heap:
Let’s Try!

- Insert 5, 3, 2, 6, 0 into a Min binary heap
Insertion into a Max Binary Heap

Algorithm:
indexOfRoot = 0
indexOfBottom = elementCount
insert new element @ “bottom” of heap (@ indexOfBottom)
elementCount ++
reHeapUp( indexOfBottom )

// If the element has a parent and ...
if (indexOfBottom > indexOfRoot)
    indexOfParent = floor((indexOfBottom - 1) / 2)
    // ... key value of parent < key value of child then ...
    if (heap[indexOfParent] < heap[indexOfBottom])
        //... swap the element with its parent
        swap heap[indexOfParent] with heap[indexOfBottom]
        reHeapUp( indexOfParent )
Insertion – Let’s Try!

Insert ____ in Max binary heap:
Insertion – Let’s Try!

Insert ____ in Max binary heap:
Let's Try!

- Insert 5, 3, 2, 6, 0 into a Max binary heap
Deletion from a Min Binary Heap

Algorithm:

indexOfRoot = 0
Call retrieve() (optional)

-> return element stored in root of heap (array[indexOfRoot])

Replace element stored in root with element stored at the bottom of heap i.e., last element

-> copy element stored at the bottom (at index elementCount-1) into root of heap

-> elementCount --

reHeapDown( indexOfRoot )

if ( heap[indexOfRoot] is not a leaf )

set indexOfMinChild to index of smallest child of root

if ( heap[indexOfRoot] > heap[indexOfMinChild] )

swap heap[indexOfRoot] with heap[indexOfMinChild]

reHeapDown( indexOfMinChild )
Deletion – Let’s Try!

indexOfRoot:

indexOfMinChild:
Deletion from a Max Binary Heap

Algorithm:

indexOfRoot = 0
Call retrieve( ) (optional)
  -> return element stored in root of heap (array[indexOfRoot])
Replace element stored in root with element stored at the bottom of heap i.e., last element
  -> copy element stored at the bottom (at index elementCount-1) into root of heap
  -> elementCount --
reHeapDown( indexOfRoot )
  if ( heap[indexOfRoot] is not a leaf )
    set indexOfMaxChild to index of largest child of root
    if ( heap[indexOfRoot] < heap[indexOfMaxChild] )
      swap heap[indexOfRoot] with heap[indexOfMaxChild]
      reHeapDown( indexOfMaxChild )
Deletion – Let’s Try!

indexOfRoot:

indexOfMaxChild:
Time Efficiency of Binary Heap

- Insertion and Deletion ->
  - During ReHeapUp and ReHeapDown:
    - Visit each level of heap
    - Compare parent with child and swap if necessary
    - Therefore:

- Retrieve ->
  (Remember: Retrieve returns the element located at the root - Minimum or maximum)

- getElementCount ->
What about Traverse?

- Can we efficiently traverse “in order” a heap as we do with a BST or an AVL?
- Why?
Traversing “in order” a Binary Heap

Considering a node in the binary heap, no sort ordering constraint dictates the position of its children. The only sort ordering constraint that exist in a binary heap is the rule that dictates the position of a child versus its parent.
Forte of Binary Heaps

- Finding the element with the largest/smallest key value
Summary

- Defined binary heaps
- Demonstrated and traced their recursive operations
Next Lecture

- Heap Sort