Data Structures and Algorithms

A.Baskar

BITS Pilani, K. K. Birla Goa Campus

February 26, 2018

Recap

- Non linear data structures: Trees
- Tree ADT

Tree Abstract Data Type

- Tree ADT stores elements at nodes
- element(v) returns the object stored at the node v $O(1)$
- size(), root(), parent(v)$O(1)$
- children(v) $O(c_v)$
- isInternal(v), isExternal(v), IsRoot(v) $O(1)$
- elements(), positions() $O(n)$
- swapElements(v,w), replaceElements(v,e) $O(1)$

Outline

- Heaps
- Heap sort
- Priority Queue ADT

Heaps

- Storing elements and keys in internal nodes of binary tree
- External nodes will not have any element
- Heap-order property and complete binary tree property

Heap-order property

- Every node v other than the root, the key stored at v is greater than or equal to the key stored at v's parent.

Heap-order property

- Every node v other than the root, the key stored at v is greater than or equal to the key stored at v's parent.
- Keys on path from root node to an external node are in nondecreasing order
- The root node will have the minimum key

Complete binary tree

- A binary tree is a complete binary tree if in every level, except possibly the deepest, is completely filled. At depth n, the height of the tree, all nodes must be as far left as possible.

Complete binary tree

- A binary tree is a complete binary tree if in every level, except possibly the deepest, is completely filled. At depth n, the height of the tree, all nodes must be as far left as possible.
- There is a special node called last node
- A heap storing n keys has height $\lceil log(n+1) \rceil$

Array representation of a heap

- We can use array to represent heap and index of last node is equal to n
- If there are n keys to be stored, there will be $2n + 1$ nodes in the tree
- But it is not necessary to store all of them in the array representation

Insertion in a heap

- If we want to insert a key k in the heap, first we have to identify the correct external node z.
- Then we perform an expandExternal(z) operation: replaces z with an internal node (which has two external nodes)
- Then insert e at the newly created internal node.

Insertion in a heap

- If we want to insert a key k in the heap, first we have to identify the correct external node z.
- Then we perform an expandExternal(z) operation: replaces z with an internal node (which has two external nodes)
- Then insert e at the newly created internal node.
- It might violate the heap-order property
- Up-heap bubbling to resolve this issue
- Insert method takes O(logn) time

removeMin in a heap

- First copy the key in the last node to root node
- Now change the last node as external node
- Reassign the last node

removeMin in a heap

- First copy the key in the last node to root node
- Now change the last node as external node
- Reassign the last node
- It might violate the heap-order property
- Down-heap bubbling to resolve this issue
- removeMin method takes O(logn) time

Heap-sort

- First we have to insert n items and it will take $O(nlogn)$ time
- Then we have to removeMin n times and it will take $O(nlogn)$ time
- So overall running time for heap-sort is $O(nlogn)$

Priority Queue ADT

- In selection sort, insertion sort and heap sort, we are using only a few methods
- removeMin, removeFirst, insert

Priority Queue ADT

- In selection sort, insertion sort and heap sort, we are using only a few methods
- removeMin, removeFirst, insert
- In selection sort and heapsort we use removeMin and insert
- In insertion sort we use removeFirst and insert

Priority Queue ADT

- In selection sort, insertion sort and heap sort, we are using only a few methods
- removeMin, removeFirst, insert
- In selection sort and heapsort we use removeMin and insert
- In insertion sort we use removeFirst and insert
- Priority Queue ADT supports these methods

Priority Queue Implementation: Unsorted sequence

- Using unsorted sequence to implement priority queue
- insert method takes constant time
- removeFirst and removeMin take O(n) time

Priority Queue Implementation: Unsorted sequence

- Using unsorted sequence to implement priority queue
- insert method takes constant time
- removeFirst and removeMin take O(n) time
- Selection sort can be seen as Priority Queue sort as follows
- From a given collection C insert element into the Priority Queue Q
- Use removeMin on Q and store it in C

Priority Queue Implementation: Sorted sequence

- Using sorted sequence to implement priority queue
- removeMin method takes constant time
- insert method takes O(n) time

Priority Queue Implementation: Sorted sequence

- Using sorted sequence to implement priority queue
- removeMin method takes constant time
- insert method takes O(n) time
- Insertion sort can be seen as Priority Queue sort as follows
- From a given collection C insert element into the Priority Queue Q
- Use removeMin on Q and store it in C

Priority Queue Implementation: Heap Tree

- Both the above implementations use linear data structure
- A nonlinear data structure to implement the priority queue in an efficient way?

Priority Queue Implementation: Heap Tree

- Both the above implementations use linear data structure
- A nonlinear data structure to implement the priority queue in an efficient way?
- In heap implementation insert and removeMin methods take O(logn) time
- From a given collection C insert element into the Priority Queue Q
- Use removeMin on Q and store it in C

Priority Queue Implementation: Heap Tree

- Both the above implementations use linear data structure
- A nonlinear data structure to implement the priority queue in an efficient way?
- In heap implementation insert and removeMin methods take O(logn) time
- From a given collection C insert element into the Priority Queue Q
- Use removeMin on Q and store it in C
- This is known as heap sort and it takes O(nlogn) time