Data Structures and Algorithms Design

A.Baskar

BITS Pilani, K. K. Birla Goa Campus

26 September 2015

Recap

- Non linear data structures: Trees
- Tree terminology
- Binary trees
- Tree ADT, and array implementation

Convention

- All the trees are rooted and ordered
- Edges have direction, from above to below
- All the binary trees are proper

Tree Abstract Data Type

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

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)

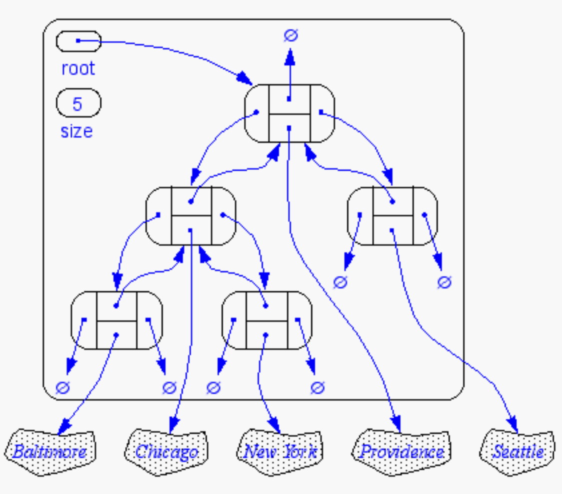
Outline

- Binary Tree linked list representation
- Tree traversal
- Sorting problem
- Insertion sort, Selection sort
- Heaps
- Heap sort

Binary Trees: Linked List Representation

- We use linked list to represent binary trees
- Each node v will be associated with an object with references
  - to the element stored at v
  - the positions associated with the children
  - the position associated with the parent

Binary Trees: Linked List Representation



Tree Abstract Data Type

- 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)

Tree traversal

- A binary tree is defined recursively: it consists of a root, a left subtree and a right subtree
- To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once.
- Tree traversals are naturally recursive.

Tree traversal

- A binary tree is defined recursively: it consists of a root, a left subtree and a right subtree
- To traverse (or walk) the binary tree is to visit each node in the binary tree exactly once.
- Tree traversals are naturally recursive.
- Since a binary tree has three parts, there are six possible ways to traverse the binary tree:
- root, left, right : preorder
- left, root, right: inorder
- left, right, root: postorder

Algorithms

- Tree traversal algorithms
- Finding depth of a node in a tree
- Finding height of the tree T

Sorting Problem

- Input : A sequence of n numbers $< a_1, a_2, \ldots, a_n >$
- Output : A permutation (reordering) $< a'_1, a'_2, \ldots, a'_n >$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

Sorting Problem

- Input : A sequence of n numbers $< a_1, a_2, \ldots, a_n >$
- Output : A permutation (reordering) $< a'_1, a'_2, \ldots, a'_n >$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$
- Solutions : Many!
- First Solution : Selection Sort

Selection Sort

- Start with an empty left hand and the cards face up on the table
- Find the minimum card on the table and insert it as a last card in the left hand
- Not much work during insertion but only during selection
- Finding minimum needs O(n) time

Insertion Sort

- Inserting an element into a sorted list in the appropriate position retains the order.
- Works the way many people sort a hand of playing cards.
- Start with an empty left hand and the cards face down on the table.
- We remove one card from the table and insert it in the correct position in left hand

Insertion Sort

- Inserting an element into a sorted list in the appropriate position retains the order.
- Works the way many people sort a hand of playing cards.
- Start with an empty left hand and the cards face down on the table.
- We remove one card from the table and insert it in the correct position in left hand
- To find the correct position for a card, we compare it with each of the cards already in the hand, from right to left.

Insertion Sort

- Inserting an element into a sorted list in the appropriate position retains the order.
- Works the way many people sort a hand of playing cards.
- Start with an empty left hand and the cards face down on the table.
- We remove one card from the table and insert it in the correct position in left hand
- To find the correct position for a card, we compare it with each of the cards already in the hand, from right to left.

Remark

At all times the cards in the left hand are sorted, and these cards were originally the top cards of the pile on the table.

Analysis

- The best case for insertion sort occurs when the list is already sorted. In this case, insertion sort requires n-1 comparisons
- Best case analysis: $O(n)$ complexity.

Analysis

- The best case for insertion sort occurs when the list is already sorted. In this case, insertion sort requires n-1 comparisons
- Best case analysis: $O(n)$ complexity.
- We consider the worst case analysis
- For each value of i, what is the maximum number of key comparisons possible?

Analysis

- The best case for insertion sort occurs when the list is already sorted. In this case, insertion sort requires n-1 comparisons
- Best case analysis: $O(n)$ complexity.
- We consider the worst case analysis
- For each value of i, what is the maximum number of key comparisons possible?
- Answer: i -1

Analysis

- The best case for insertion sort occurs when the list is already sorted. In this case, insertion sort requires n-1 comparisons
- Best case analysis: $O(n)$ complexity.
- We consider the worst case analysis
- For each value of i, what is the maximum number of key comparisons possible?
- Answer: i -1
- Thus, the total time in the worst case is $O(n^2)$

Priority Queue ADT

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

Priority Queue ADT

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

Priority Queue ADT

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

Heaps

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

Heaps

- Both the above implementations use linear data structure
- Can we use some 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

Heaps

- Both the above implementations use linear data structure
- Can we use some 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

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.