21BDS0340

Abhinav Dinesh Srivatsa

Data Structures and Algorithms

Assignment – II

1. Operations of BST:

Insertion: Add element to tree

Deletion: Delete an element from tree

Search: Find an element in the tree

Inorder traversal: Display tree elements in the order of left, root, right

Preorder traversal: Display root, left, right

Postorder traversal: Display left, right, root

```
insert(n, node):
    If node is null
        Assign node's value as n and return
    If  n > node's value
        call insert(n, node.right)
    If  n ≤ node's value
        call insert(n, node.left)
```

```
search(n, node):
    If node = null, then return (not found)
    If n > node, call search(n, node.right)
    Else if n < node, call search(n, node.left)
    Else return that element is found
```

delete(n, node):

If node's value = n

If nodes has 0 children, then delete the node

If node has 1 child, then assign node's child to node's parent

If nodes has 2 children, then replace it with the element at the leftmost of the right subtree (next successor)

Else if n > node's value, then call delete(n, node.right)

Else, call delete(n, node.left)

inorder(node):

If node is null, return

~~Print no~~

call inorder(node.left)

Print node's value

call inorder(node.right)

preorder(node):

If node is null, return

~~call~~

Print node's value

call preorder(node.left)

call preorder.(node.right)

postorder (node):

If node is null, return

Call postorder(node.left)

Call postorder(node.right)

Print node's value

2. Full Binary Tree

Every node has 0 or 2 children

Also known as a proper binary tree

Complete Binary Tree

Every level is completely filled

2^n elements at level n

Last level has leaf nodes leaning left only

Perfect Binary Tree

Every level must have 2^n children at level n

All nodes internal must have 2 children

Balanced Binary (AVL) Tree

Difference between left and right subtrees

can be at most 1.

Height is $O(\log_2 n)$ where n is no. of elements

3. Linear search is going through an array one index at a time to check if the element corresponds

Ex:

2 4 0 1 9

search for 1

2 4 0 1 9
↑
x

2 ≠ 1

2 4 0 1 9
  ↑
  x

4 ≠ 1

2 4 0 1 9
    ↑
    x

0 ≠ 1

2 4 0 1 9
      ↑
      x

1 = 1

Element found.

Binary search works on sorted algorithms by using the divide and conquer method.

Ex:

21 34 43 57 66 78

search for 66

21 34 43 57 66 78

↑

x

57 < 66

57 66 78

↑

x

66 = 66

Element found

4. Lists are used for applications with collections of data. A list is organised by nodes.

A node can contain a value and a pointer pointing to the next node. Combined, the nodes make a list.

Applications of lists include polynomial manipulation, implementing stacks and queues.

5. Selection sort

The min value is swapped with the initial unsorted element.

Ex:

16 23 11 46 98

Min = 11

11 23 16 46 98

Min = 16

11 16 23 46 98

Min = 23, no change

Min = 46, no change

Min = 98, no change

Counting Sort

This sort is implemented by counting the amount of elements in the array and seeing where the sum values change.

Ex:

4 5 1 2 8 9 3

counts:

0 1 1 1 0 1 1 0 1 1

0 9

Summing from left:

0 1 2 3 3 4 5 5 6 6

sorted array: (indices)

1 2 3 5 6 8 9

Quick Sort

This sort is implemented with a divide and conquer method of splitting the elements to two groups of lesser and greater than a pivot.

Ex:

36 29 84 93 56 19

Let 36 be pivot

```
36  29  84  93  56  19
↑                   ↑
P                   R
```

P > R, swap

```
19  29  84  93  56  36
    ↑               ↑
    L               P
```

P > L, no swap

```
19  29  84  93  56  36
        ↑           ↑
        L           P
```

P < L, swap

```
19  29  36  93  56  84
        ↑       ↑
        P       R
```

P < R, no swap

```
19  29  36  93  56  84
        ↑   ↑
        P   R
```

P < R, no swap

Left part:

```
19  29
↑   ↑
P   R
```

P < R, no swap

sorted

Right part:

```
93  56  84
↑       ↑
P       R
```

P > R, swap

```
84  56  93
    ↑   ↑
    L   P
```

L < P, no swap

Left part:

```
84  56
↑   ↑
P   R
```

P > R, swap

```
56  84
↑
P
```

Putting together:

19 29 36 56 84 93