DS.T.1

Trees

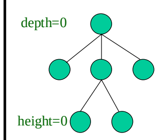
Chapter 4 Overview

- **Tree Concepts**
- **Traversals**
- **Binary Trees**
- **Binary Search Trees**
- **AVL Trees**
- **Splay Trees**
- **B-Trees**

DS.T.2

Terminology

Trees are hierarchic structures.



- root
- leaves
- parent
- children
- ancestors
- descendants
- path
- path length
- depth / level
- height
- subtrees

DS.T.3

Recursive Definition:
A tree is a set of nodes that is
a. empty or
b. has one node called the root from which zero or more trees descend.

General (n-ary) Arithmetic Expression Tree

(A + B + ((C * D * E) / F) + G) - H

How can we implement general trees with whose nodes can have variable numbers of children?

DS.T.4

Common Traveral Orders for General Trees

- Preorder
- Postorder

```
void print_preorder ( TreeNode T)
{
TreeNode P;
if  ( T == NULL )      return;
else  {
        print T-> Element;
        P = T -> FirstChild;
        while (P != NULL)
          {
          print_preorder ( P );
          P = P -> NextSibling;
          }
        }
}
```

DS.T.5

A **binary tree** is a tree in which each node has two subtrees--left and right.

Either or both may be empty.

(A + B + ((C * D * E) / F) + G) - H

What operations are required for a binary tree?
That depends …

DS.T.6

- Binary Arithmetic Expression Trees

  - construct from infix expression
  - add or delete nodes
  - traverse in preorder to produce prefix expression
  - traverse in postorder to evaluate
  - traverse in inorder to output infix expression

- Binary Decision Trees

- Binary Search Trees

DS.T.7

Recursive Preorder Traversal

```
void RPT (TreeNode T)
{
if (T != NULL) {
  "process" T -> Element;


}
```

Preorder Traversal with a Stack

```
void SPT (TreeNode T, Stack S)
{
if (T == NULL) return; else push(T,S);
while (!isempty(S)) {
  T = pop(S);
  "process" T -> Element;
  if (T -> Right != NULL) push(T -> Right, S);
  if (T -> Left  != NULL) push(T -> Left, S);
  }
}
```

DS.T.8

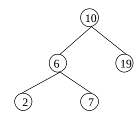
Binary Search Trees

Search trees are look-up tables that are used to find a given key value and return associated data.

Example: look up SSN, return name and address.

A binary tree satisfies the ordering property if the key value in any given node is

> all key values in the node's left subtree
≤ all key values in the node's right subtree



DS.T.9

Operations

- Find the node with a given key
- FindMin / FindMax key in the tree
- Insert a new key (and associated data)
- Delete a key (and associated data)

Find, FindMin, FindMax, Insert are easy.

Delete is a little bit tricky.

DS.T.10

Deletion of a Node from a Binary Search Tree

1. Find the node with the given key value.

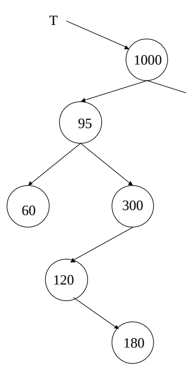
2. Delete that node from the tree.

Problem: When you delete a node, what do you replace it by?

- If it has no children, by NULL.
- If it has one child, by that child.
- If it has two children, by the node with the smallest key in its right subtree.

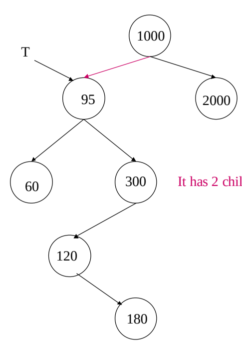
DS.T.11

Delete node with key 95.



DS.T.12

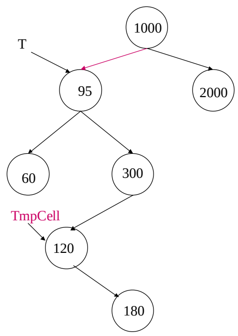
Find the node.



It has 2 children.

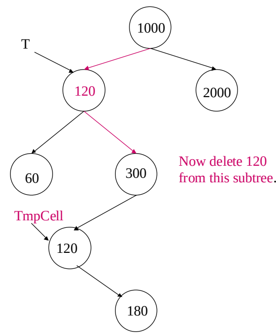
DS.T.13

Find smallest in its right subtree.



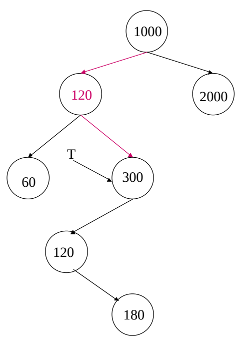
DS.T.14

Replace T's key value with that of TmpCell.



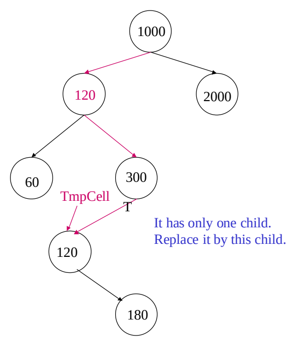
DS.T.15

Delete 120 from this subtree T.

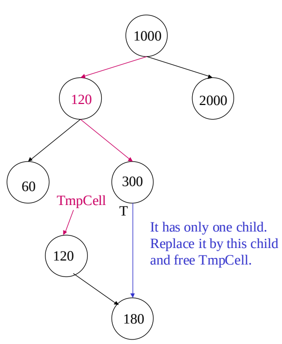


DS.T.16

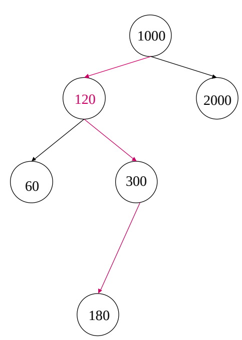
Find the 120 again.



DS.T.17



DS.T.18



DS.T.19

What do you think of this delete procedure?

Is it readable?

Is it efficient?

How would YOU do it?